I, Murthy N. Revanuru, hereby submit this as part of the requirements for the degree of:

Master of Science

in:

Computer Engineering

It is entitled:

"Modeling and Simulation of an FM Receiver using VHDL-AMS"

Approved by:

Dr. Harold W. Carter (chair)
Dr. Joseph H. Nevin
Dr. Fred Beyette
Modeling and Simulation of an FM Receiver using VHDL-AMS

A thesis submitted to the

Division of Graduate Studies and Research of
the University of Cincinnati

in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in the Department of
Electrical and Computer Engineering and Computer Science
of the College of Engineering

March 2001

by

Murthy N. Revanuru
B.Tech., (EEE), Jawaharlal Nehru Technological University,
Hyderabad, India, 1998.

Thesis Advisor and Committee Chair: Dr. Harold W. Carter
Abstract

The limitations of the SPICE language have given rise to new simulation languages and environments. VHDL-AMS is one such language, which supports analog and mixed-mode designs. Its unique features like Ordinary Differential Algebraic Equations (ODAE’s), discontinuity identification and processing allow it to model systems over a wide rage of physical domains. VHDL-AMS allows the description of any system at whatever level of abstraction is appropriate for a given simulation accuracy-Versus- time tradeoff.

This thesis investigates the feasibility and performance of VHDL-AMS towards the time-domain modeling of systems in the Radio Frequency region. In particular, the performance of VHDL-AMS in modeling an FM receiver is analyzed and its applicability issues are studied. The VHDL-AMS models are simulated using SEAMS, a mixed signal simulator under development at the Distributed Processing laboratory, University of Cincinnati.
To my loving parents
And
my dearest brother
Acknowledgements

I would like to thank Dr. Carter for his support and guidance, without which this work would not have been possible. My sincere thanks to Dr. Nevin and Dr. Beyette for readily accepting to serve on my thesis committee.

A special thanks goes to my parents and my brother for their constant support and encouragement.

I would also like to thank all my friends at U.C for all the good times we had together.
# Contents

## List Of Figures

## List of Tables

## Introduction

1.1 Motivation.............................................................................................................. 1  
1.2 Problem Statement ................................................................................................. 2  
1.3 Approach................................................................................................................ 2  
1.4 Results....................................................................................................................2  
1.5 Overview of the document..................................................................................... 4

## Background

2.1 Modeling Requirements of RF Systems ................................................................. 5  
2.2 Applicability of VHDL-AMS ................................................................................ 6  
2.3 Suggested Extensions for VHDL-AMS ................................................................. 7  
2.4 Frequency Modulation and Receivers ................................................................... 7  
2.4.1 Frequency Modulation (FM) Wave....................................................................... 8  
2.4.2 Receiver ............................................................................................................ 10  
2.4.3 Receiver ............................................................................................................ 11  
2.4.4 Superheterodyne FM Receiver Configurations................................................. 11  
2.5 Related Research .................................................................................................... 11

## VHDL – AMS MODELS

3.1 Guidelines ............................................................................................................ 14  
3.2 Different Abstraction levels .................................................................................. 15  
3.3 Component Modeling .......................................................................................... 16  
3.3.1 Mixer ................................................................................................................. 16  
3.3.1.1 Mixer Action.................................................................................................. 17  
3.3.1.2 Passive Mixer............................................................................................... 17  
3.3.1.3 Active Mixer ............................................................................................... 18  
3.3.2 Limiter .............................................................................................................. 20  
3.3.3 FM Discriminator (Demodulator) .................................................................... 22  
3.3.4 Filter ................................................................................................................. 25  
3.3.5 Voltage controlled Oscillator ......................................................................... 29  
3.3.6 Amplifier ......................................................................................................... 30
Performance Evaluation

4.1 Criteria ................................................................................................................ 33
4.1.1 Error Criteria ................................................................................................. 33
4.1.2 Execution Time Criteria ................................................................................ 34
4.2 Simulation Environment ...................................................................................... 35
4.3 Performance Evaluation ....................................................................................... 35
4.3.1 Limiter ........................................................................................................... 36
4.3.2 Mixer ............................................................................................................. 41
4.3.3 RF Amplifier ................................................................................................. 46
4.3.4 Filters ............................................................................................................. 50
4.2.5 FM Detector .................................................................................................. 57
4.3.6 IF Amplifier ................................................................................................... 60
4.3.7 FM Receiver .................................................................................................. 61

Conclusions and Future Work                                                                                        67

5.1 Summary .............................................................................................................. 67
5.2 Future Work ......................................................................................................... 68

Bibliography                                                                                                                     69

Appendix

VHDL-AMS CODE                                                                                                         71

A.1 Signal Generators................................................................................................ 71
A.1.1 Sine Wave .................................................................................................... 71
A.1.2 Square Wave ................................................................................................ 71
A.1.3 Ramp Wave .................................................................................................. 72
A.1.4 FM Wave ...................................................................................................... 72
A.1.5 AM Wave ..................................................................................................... 73
A.2. Limiter................................................................................................................. 74
A.2.1 Conceptual Level Model .............................................................................. 74
A.2.2 Behavioral Level Model............................................................................... 74
A.2.3 Structural Level Model................................................................................. 75
A.3 Mixer................................................................................................................... 77
A.3.1 Conceptual Model ........................................................................................ 77
A.3.2 Behavioral Model ......................................................................................... 77
A.3.3 Structural Model........................................................................................... 79
A.3.4 Active Mixer Using a FET ........................................................................... 80
A.4 RF Amplifier....................................................................................................... 82
A.4.1 Conceptual Level.......................................................................................... 82
A.4.2 Structural Model of a 2-stage amplifier ....................................................... 82
A.5 IF Amplifier ........................................................................................................ 84
A.6 FM Detector ...................................................................................................... 85
A.6.1 Travis Detector ............................................................................................. 85
A.6.2 Slope Detector .............................................................................................. 86
A.7 Filter.................................................................................................................... 87
A.7.1 Structural Level model of a Active LPF using ideal op-amp....................... 87
A.7.2  Structural Level model of a Active HPF using ideal op-amp ...................... 88
A.7.3  Structural Level model of a Active BPF using ideal op-amp ...................... 89
A.7.4  Structural Level model of a Active LPF using op-amp 741 ........................ 90
A.7.5  Structural Level model of a Active HPF using op-amp 741 ..................... 91
A.8  Voltage Controlled Oscillator ............................................................................. 93
A.9  Oscillator ............................................................................................................. 93
A.10  FM-to-AM Converter ......................................................................................... 95
A.11  Transformers ....................................................................................................... 96
A.11.1 Untuned Transformer .................................................................................... 96
A.11.2 Untuned Secondary Tapped Transformer ..................................................... 96
A.11.3 Double Tuned Transformer ......................................................................... 97
A.11.4 Double Tuned Secondary Tapped Transformer ............................................ 98
A.12  Operational Amplifier ....................................................................................... 100
A.12.1 Simple Operational Amplifier ..................................................................... 100
A.12.2 Simple Operational Amplifier(Using 5 BJT’s) ........................................... 100
A.12.3 Operational Amplifier(L741) ..................................................................... 102
A.13  Crystal ............................................................................................................... 105
A.14  Frequency Doubler ............................................................................................. 106
A.15  AM Detector ...................................................................................................... 108
A.16  Transistors ......................................................................................................... 111
A.16.1 Trans_T_Equivalent Structural Model ..................................................... 111
A.16.2 Behavioral Model of NPN_Transistor ......................................................... 112
A.16.3 Behavioral Model of PNP_Transistor ......................................................... 114
A.17  Structural Model of the FET ............................................................................. 115
A.18  Diode .................................................................................................................. 116
A.19  Resistor ............................................................................................................. 117
A.20  Capacitor ........................................................................................................... 117
A.21  Inductor ............................................................................................................. 118
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>FM carrier signal</td>
<td>10</td>
</tr>
<tr>
<td>2.2</td>
<td>Block Diagram of a FM Receiver</td>
<td>12</td>
</tr>
<tr>
<td>3.1</td>
<td>Double Balanced, diode ring Mixer</td>
<td>17</td>
</tr>
<tr>
<td>3.2</td>
<td>A Simple JFET Mixer</td>
<td>18</td>
</tr>
<tr>
<td>3.3</td>
<td>Limiter Input/Output Relationship</td>
<td>20</td>
</tr>
<tr>
<td>3.4</td>
<td>Limiter Circuit Diagram</td>
<td>21</td>
</tr>
<tr>
<td>3.5</td>
<td>Travis detector (Slope Detector) circuit</td>
<td>23</td>
</tr>
<tr>
<td>3.6</td>
<td>Output Characteristics of the travis detector</td>
<td>24</td>
</tr>
<tr>
<td>3.7.1</td>
<td>A Passive Low Pass Filter</td>
<td>25</td>
</tr>
<tr>
<td>3.7.2</td>
<td>A Passive High Pass Filter</td>
<td>25</td>
</tr>
<tr>
<td>3.8.1</td>
<td>Low Pass Filter Using an OP-AMP</td>
<td>26</td>
</tr>
<tr>
<td>3.8.2</td>
<td>High Pass Filter Using an OP-AMP</td>
<td>26</td>
</tr>
<tr>
<td>3.9</td>
<td>Trans T-equivalent Model of the HF transistor</td>
<td>30</td>
</tr>
<tr>
<td>3.10</td>
<td>Two stage Untuned Amplifier</td>
<td>31</td>
</tr>
<tr>
<td>3.11</td>
<td>Single Stage Tuned Amplifier</td>
<td>32</td>
</tr>
<tr>
<td>4.1</td>
<td>Block Diagram of a FM receiver</td>
<td>36</td>
</tr>
<tr>
<td>4.2</td>
<td>Testbench for the limiter circuit</td>
<td>37</td>
</tr>
<tr>
<td>4.3</td>
<td>Output from the structural model of the limiter</td>
<td>37</td>
</tr>
<tr>
<td>4.4</td>
<td>Absolute Error of the structural model of the limiter</td>
<td>38</td>
</tr>
<tr>
<td>4.5</td>
<td>Output from the behavioral model of the limiter</td>
<td>39</td>
</tr>
<tr>
<td>4.6</td>
<td>Absolute Error for the behavioral model of the Limiter</td>
<td>39</td>
</tr>
<tr>
<td>4.7</td>
<td>Output from the conceptual model of the Limiter</td>
<td>47</td>
</tr>
<tr>
<td>4.8</td>
<td>Absolute Error for the conceptual model of the Limiter</td>
<td>47</td>
</tr>
<tr>
<td>4.9</td>
<td>Testbench for the mixer circuit</td>
<td>41</td>
</tr>
<tr>
<td>4.10</td>
<td>Output from the structural model of the Mixer</td>
<td>42</td>
</tr>
<tr>
<td>4.11</td>
<td>Absolute Error for the structural model of the Mixer</td>
<td>42</td>
</tr>
<tr>
<td>4.12</td>
<td>Output from the behavioral model of the Mixer</td>
<td>43</td>
</tr>
<tr>
<td>4.13</td>
<td>Absolute Error for the behavioral model of the Mixer</td>
<td>44</td>
</tr>
<tr>
<td>4.14</td>
<td>Output from the conceptual model of the Mixer</td>
<td>45</td>
</tr>
<tr>
<td>4.15</td>
<td>Absolute Error for the conceptual model of the Mixer</td>
<td>45</td>
</tr>
<tr>
<td>4.16</td>
<td>Testbench for the RF Amplifier</td>
<td>47</td>
</tr>
<tr>
<td>4.17</td>
<td>Output from the structural/behavioral model of the RF Amplifier</td>
<td>47</td>
</tr>
<tr>
<td>4.18</td>
<td>Absolute Error for the struct/behav model of the RF Amplifier</td>
<td>48</td>
</tr>
<tr>
<td>4.19</td>
<td>Output from the conceptual model of the RF Amplifier</td>
<td>49</td>
</tr>
<tr>
<td>4.20</td>
<td>Absolute Error for the conceptual model of the RF Amplifier</td>
<td>49</td>
</tr>
<tr>
<td>Figure 4.21</td>
<td>Testbench for the filter circuit ........................................................................... 51</td>
<td></td>
</tr>
<tr>
<td>-------------</td>
<td>----------------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>Figure 4.22</td>
<td>Output from the struct/behav model of the LPF .................................................... 51</td>
<td></td>
</tr>
<tr>
<td>Figure 4.23</td>
<td>Absolute Error for the struct/behav model of the LPF ........................................... 52</td>
<td></td>
</tr>
<tr>
<td>Figure 4.24</td>
<td>Frequency Response of a Low Pass Filter ............................................................... 53</td>
<td></td>
</tr>
<tr>
<td>Figure 4.25</td>
<td>Output from the struct/behav model of the HPF ...................................................... 53</td>
<td></td>
</tr>
<tr>
<td>Figure 4.26</td>
<td>Absolute Error for the struct/behav model of the HPF ............................................ 54</td>
<td></td>
</tr>
<tr>
<td>Figure 4.27</td>
<td>Frequency Response of a High Pass Filter ............................................................. 55</td>
<td></td>
</tr>
<tr>
<td>Figure 4.28</td>
<td>Output from the struct/behav model of the BPF ...................................................... 56</td>
<td></td>
</tr>
<tr>
<td>Figure 4.29</td>
<td>Absolute Error for the struct/behav model of the BPF ............................................ 56</td>
<td></td>
</tr>
<tr>
<td>Figure 4.30</td>
<td>Frequency Response of the BPF .............................................................................. 57</td>
<td></td>
</tr>
<tr>
<td>Figure 4.31</td>
<td>Testbench for the travis detector ............................................................................ 58</td>
<td></td>
</tr>
<tr>
<td>Figure 4.32</td>
<td>Output from the struct/behav model of the FM Detector ......................................... 58</td>
<td></td>
</tr>
<tr>
<td>Figure 4.33</td>
<td>Output from the struct/behav model of the FM Detector ......................................... 59</td>
<td></td>
</tr>
<tr>
<td>Figure 4.35</td>
<td>Frequency response of the IF Amplifier .................................................................. 60</td>
<td></td>
</tr>
<tr>
<td>Figure 4.36</td>
<td>Testbench for the FM Receiver (part 1) .................................................................... 61</td>
<td></td>
</tr>
<tr>
<td>Figure 4.37</td>
<td>Output from the part 1 of the FM receiver .............................................................. 62</td>
<td></td>
</tr>
<tr>
<td>Figure 4.38</td>
<td>Testbench for the FM Receiver (part 2) .................................................................... 62</td>
<td></td>
</tr>
<tr>
<td>Figure 4.39</td>
<td>Output from the part 2 of the FM receiver .............................................................. 63</td>
<td></td>
</tr>
</tbody>
</table>
List of Tables

Table 2.1     RF Modeling Requirements ................................................................. 6
Table 3.1     Circuit parameters of the travis detector .............................................. 24
Table 4.1     SEAMS execution times ................................................................. 64
Table 4.2     SPICE execution times ................................................................. 65
Table 4.3     Performance of various limiter models .............................................. 41
Table 4.4     Performance of various mixer models .............................................. 46
Table 4.5     Performance of various RF Amplifier models .................................. 50
Table 4.6     Relative error of all models in dB .................................................. 66
Chapter 1

Introduction

1.1 Motivation

For more than two decades, the analog community has been relying on variations of the original Berkeley SPICE, as the simulation tool for verifying and fine-tuning analog designs. The limitations of the SPICE language has given rise to new simulation languages and environments. VHDL-AMS is one such language, which supports analog and mixed-mode designs. Its unique features like Ordinary Differential Algebraic Equations (ODAE’s), discontinuity identification and processing allow it to model systems over a wide range of physical domains. VHDL-AMS allows the description of any system at whatever level of abstraction is appropriate for a given simulation accuracy-Versus-time tradeoffs.

This thesis exploits the rich features of VHDL-AMS like the break statement and laplace transform function (ltf) to model systems in the Radio Frequency (RF) region and investigates its performance and applicability.

---

1 Simulation Program with Integrated Circuit Emphasis
2 VHSIC (Very High Speed Integrated Circuit) Hardware Description Language with Analog and Mixed Signal extensions
1.2 Problem Statement

This thesis investigates the feasibility and performance of VHDL-AMS towards the time-domain modeling of systems in the Radio Frequency region. In particular, the performance of VHDL-AMS in modeling an FM Receiver is analyzed and its applicability issues are studied. The work performed in this thesis can be divided into two major tasks:

1. Develop efficient VHDL-AMS models for each of the sub-systems at various abstraction levels.
2. Investigate the performance of VHDL-AMS when modeling systems in the Radio Frequency region.

1.3 Approach

The approach adopted in this thesis is to develop effective models in VHDL-AMS, execute them on SEAMS (Simulation Environment for Analog and Mixed Signals), the mixed signal simulator developed in our laboratory (Distributed Processing Laboratory, UC) and validate the results using SPICE. Each of the FM receiver sub-systems (discussed in Chapter 2) is modeled at the three abstraction levels (conceptual, behavioral and structural). For each subsystem, at each abstraction level, its output, simulation time and other related data is collected and then the performance study is done.

1.4 Results
The aim of this thesis was to investigate the feasibility and performance of VHDL-AMS towards modeling of systems in the high frequency region. An FM receiver was considered for modeling and analysis. The FM receiver sub-systems were modeled and its performance computed. Except for the amplifier, all the other circuits exhibit good fidelity and performance. The execution time for VHDL-AMS and SPICE models along with their relative errors is presented later in Chapter 4. The results show that VHDL-AMS is indeed capable of describing high frequency systems with reasonable accuracy. The execution time of models in SEAMS is high as compared to equivalent SPICE models. The important results obtained after using the models in the testbench circuits are summarized below:

1. LIMITER: The maximum error is –32.04 dB for the conceptual level model and the minimum error is –34.09 dB for the behavioral level model.
2. MIXER: The maximum error is –10.76 dB for the conceptual level model and the minimum error is –26.56 dB for the behavioral level model.
3. LOW PASS FILTER: The maximum error is –36.79 dB for the conceptual level model and the minimum error is –41.23 dB for the structural and behavioral level models.
4. HIGH PASS FILTER: The maximum error is –35.98 dB for the conceptual level model and the minimum error is –42.05 dB for the structural and behavioral level models.
5. BAND PASS FILTER: The error is –42.84 dB for the structural and behavioral level models.
6. RF AMPLIFIER: The maximum error is –29.84 dB for the conceptual level and the minimum error is –38.97 dB for the structural and behavioral level models (the primitive high frequency transistor model is used).
7. TRAVIS DETECTOR: The error is –17.41 dB for the structural and behavioral level models. The output error is mainly due to the error introduced by the interpolation technique used in error calculation.
1.5 Overview of the document

The document is organized as follows:

- **Chapter 1** introduces the topic, approach and results of the research.

- **Chapter 2** provides a concise background on FM receiver, time domain modeling of RF systems using VHDL-AMS and performance issues addressed in this research.

- **Chapter 3** discusses the various abstraction levels used in this research and provides a brief understanding on each of the receiver’s sub-systems. This chapter also presents the VHDL-AMS models for each of the sub-systems in the various abstraction levels.

- **Chapter 4** presents the criteria for evaluating the performance of models in VHDL-AMS and SPICE and later evaluates these models based on the criteria developed. This chapter also compares the performance of SEAMS with SPICE and discusses the applicability of VHDL-AMS for simulating RF systems.

- **Chapter 5** presents the conclusions of this thesis and suggestions for future work in this area of research.
Chapter 2

Background

VHDL-AMS is a unified modeling language that has been recently approved as an IEEE standard for describing systems in any discipline. In this chapter, we discuss the various requirements for modeling Radio Frequency (RF) systems [1] and how well VHDL-AMS supports them. Also discussed in this chapter are the suggested improvements to AMS (the continuous time extensions to VHDL to form VHDL-AMS) and a concise description of a typical Radio Frequency system, which we used to evaluate the capability of VHDL-AMS.

2.1 Modeling Requirements of RF Systems

Radio Frequency circuits need to be simulated in various abstraction levels and domains in-order to evaluate its performance and architecture. RF systems are best modeled in the frequency domain. To study the transients related to non-periodic effects (example: start-up), time domain modeling has to be used as no such information is available in the frequency domain. Microwave circuits also need to be modeled in the sampled time and sampled frequency domain. The modeling approach for RF systems may be linear, non-linear, Ordinary Differential Algebraic Equations (ODAE) or Partial Differential Equations (PDE) and the language must be capable of supporting these.
Noise (internal or external) in high frequency systems is always a concern and for the effective modeling of RF systems, noise sources must be supported. Another requirement of the modeling language is that it must support steady state noise analysis and local stability analysis. Some other features like s-parameter modeling, N-port modeling, sensitivity analysis, Monte Carlo analysis must be supported as well.

The modeling needs of RF systems are tabulated below in Table 1 and later we discuss how well VHDL-AMS supports each of these.

<table>
<thead>
<tr>
<th>Domain</th>
<th>Time, Frequency, Sampled time and Sampled frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Noise</td>
<td>Spectral and Additive</td>
</tr>
<tr>
<td>Modeling Methods</td>
<td>Circuit level, signal flow, transfer function, N-port, z-transform</td>
</tr>
<tr>
<td>Numerical forms</td>
<td>Linear, non-linear, ODAE and partial differential equations</td>
</tr>
</tbody>
</table>

Table 2.1: RF Modeling Requirements

### 2.2 Applicability of VHDL-AMS

VHDL-AMS does not support pure frequency domain modeling [13]. It provides some frequency resources like Laplace and Z-transfer function, but internally they are converted into their time-domain equivalent and executed. VHDL-AMS fully supports time-domain modeling for lumped-element and signal flow circuits. It does not support distributed-element modeling since the language does not support partial differential equations. In the sample frequency domain only sample-and-hold and Z domain transfer functions are supported while it fully supports sampled time domain. By the addition of a few features (described later in this chapter), it is possible to extend the applicability of VHDL-AMS to better model RF systems.

VHDL-AMS allows the modeling of a system at various levels of abstraction. Systems can be effectively modeled at the behavioral level, circuit level, or the signal flow level. Transfer functions are also supported by the language, which greatly increases
its applicability. At the same time, it does not support distributed-element modeling and n-port modeling of systems.

VHDL-AMS provides for various numerical forms for the modeling of systems like the linear, non-linear, algebraic and ODAE. Discontinuity identification and processing with the break statement allows the modeling of discontinues in a continuous time domain system. The language does not support partial differential systems.

Noise sources are fully supported by VHDL-AMS and this provides for noise analysis of the RF systems. VHDL-AMS covers almost all the essential requirements for RF system modeling. Taking all the above information into consideration, the thesis investigates the performance issues of VHDL-AMS while modeling RF systems.

2.3 Suggested Extensions for VHDL-AMS

The following suggestions are presented to extend VHDL-AMS as this provides for a better modeling language for RF systems.

- Provide partial differential equation capability so that distributed element modeling can be done.
- Provide an independent variable ‘s’ for frequency domain modeling so that n-port models can be described.
- Provide for time and frequency domain simulations irrespective of the model base.

2.4 Frequency Modulation and Receivers
This section serves to introduce the reader to the subject Frequency Modulation (FM) and receivers [2][3]. It starts of with the different types of modulation in communication systems and later provides a brief overview on the receiver architecture and subsystems.

*Modulation* of a high-frequency carrier is a process whereby some characteristic of the carrier signal is varied in accordance with the variance of another signal. This later signal is termed as the modulating signal. The three basic properties of the carrier wave are its amplitude, frequency and phase angle. Any one of the three basic characteristics of the wave can be varied in accordance with the modulating signal while the other two properties are held constant.

In *Amplitude Modulation (AM)* the frequency and phase of the carrier wave are held constant while its amplitude varies in accordance with the amplitude and frequency of the modulating signal.

In *Frequency Modulation (FM)*, the carrier amplitude is held constant while its frequency of the carrier is caused to deviate from its average or unmodulated value in accordance with the modulating signal.

In *Phase Modulation (PM)*, the amplitude of the carrier wave is held constant while its phase angle (with respect to a reference) is caused to vary in accordance with the modulating signal.

### 2.4.1 Frequency Modulation (FM) Wave

Frequency Modulation is broadly defined as the process of varying the frequency of the carrier in accordance with the modulating signal while maintaining the amplitude constant. The amount of frequency deviation above and below the center or rest frequency is a function of the amplitude-modulating signal. The amount of change or deviation of the carrier frequency is thus a measure of the audio amplitude. In commercial practice, an upper limit to the frequency deviation is set by the Federal Communications Commission (FCC) and is a compromise between two important factors: Signal-to-noise ratio and the transmission bandwidth requirement.
The deviation should be as high as possible to obtain a high signal-to-noise ratio and as low as possible to reduce the transmission bandwidth. The value chosen for commercial broadcast is 75.0 kHz, referred to as the "system deviation". A deviation of 75.0 kHz corresponds to a 100 percent modulation, while a 33.333 percent modulation corresponds to a carrier deviation of 25.0 kHz. The standard FM broadcast band consists of carrier frequencies from 88.0 MHz to 108.0 MHz, which is significantly higher than AM.

Let the unmodulated carrier wave be represented by the expression:

\[ e_c = A \sin(\omega_ct + \Phi) \]

Where,

- \( A \): Carrier amplitude
- \( \omega_c \): angular velocity of the carrier, which is a constant.
- \( \Phi \): Phase angle.
- \( e_c \): instantaneous value (of voltage or current)

The instantaneous frequency \( f \) of the FM wave is given by,

\[ F = f_c (1 + k V_m \cos(w_mt)) \]

The maximum deviation \( \delta \) is given by

\[ \delta = k V_m f_c \]

The instantaneous amplitude of the FM signal will be given by a formula of the form

\[ V = A \sin[F(\omega_c,w_m)] = A \sin(\theta) \]

In order to find \( \theta \), \( \omega \) must be integrated with respect to time. Thus,

\[ \theta = \int \omega dt = \int \omega_c (1 + k V_m \cos(w_mt))dt = \omega_c \int (1 + k V_m \cos(w_mt))dt \]

\[ = \omega_c t + (\delta f_m) \sin \omega_m t \]

Thus,

\[ V = A \sin(\omega_c t + (\delta f_m) \sin \omega_m t) \]

The modulating index for the FM, \( m_f \), is defined as
Inserting the modulating index in the above equation, we have,

\[ V = A \times \sin(\omega_c t + m_f \times \sin \omega_m t) \]

It is important to note that as the modulating frequency decreases and the modulating voltage amplitude \( \delta \) remains constant, the modulation index increases.

The FM carrier signal is shown below:

![Diagram of FM carrier signal](image)

\textit{Figure 2.1 : FM carrier signal}

### 2.4.2 Receiver

A receiver's job is to gather signals from the medium in which it exists and convert them to a form, which can be assimilated by the receiver's user. Functionally, the receiver must perform a number of tasks to achieve this. These tasks include at least:
transducer (antennas etc.) matching, selection of desired signals, rejection of undesired signals, amplification by very large values, demodulation, error detection and correction, received information conditioning and output.

2.4.3 Receiver Configurations

Many configurations have been developed for use in receiver [2], although the great preponderance of receivers today is of one general type. The remainders are divided between two other types. The three general receiver types are, respectively, superheterodyne, tuned radio frequency and regenerative. There are many variations of these three basic types, but we will concentrate only on the superheterodyne type.

Superheterodyne Receivers

In the superheterodyne receiver, the incoming signal voltage is combined with a signal generated in the receiver. This local oscillator voltage is normally converted into a signal of a higher fixed frequency. The signal at the intermediate frequency contains the same modulation as the original carrier, and is now amplified and detected to reproduce the original information. The superheterodyne receiver has the same essential components as the TRF receiver in addition to the mixer, local oscillator and intermediate frequency (IF) amplifier. A constant frequency difference is maintained between the local oscillator and the RF circuits, normally through capacitance tuning. The problems faced by the TRF receivers i.e. instability, insufficient adjacent-frequency rejection, and bandwidth variation can all be solved by the use of a superheterodyne receiver. Superheterodyne receivers outperform any other type of receiver except in a very few specialized applications.

2.4.4 Superheterodyne FM Receiver
The block diagram of a Superheterodyne FM Receiver [4] is shown in Figure 2.2. The receiver is similar to the AM receiver in many ways, but there are several differences as explained in the following paragraphs.

![Block Diagram of a FM receiver](image)

**Figure 2.2 : Block Diagram of a FM receiver**

The functionality of each of the above blocks is explained below. These blocks are the basis for the VHDL-AMS models described in chapter 3.

**RF Amplifier** This circuit amplifies any signal that lies between 88.0 MHz and 108.0 MHz. It is highly selective so that it passes only the selected carrier frequency and significant side-band frequencies that contain the audio.

**Local Oscillator** This circuit produces a sine wave at a frequency 10.7 MHz above the selected RF frequency.

**Mixer** This circuit performs the same function as in a AM receiver, except that its output is a 10.7 MHz FM signal regardless of the RF carrier frequency.

**IF Amplifier** This circuit amplifies the 10.7 MHz FM signal.
Limiter

The limiter removes any unwanted variations in the amplitude of the FM signal as it comes out of the IF amplifier and produces a constant amplitude FM output at the 10.7 MHz intermediate frequency.

Discriminator

This circuit performs the equivalent function of the detector in an AM receiver and is often called a detector. This discriminator recovers the audio from the FM signal.

De-emphasis Net.

The de-emphasis network in the FM receiver brings the high-frequency audio signals back to the proper amplitude relationship with the lower frequencies.

Audio Amplifiers

This circuit is the same as in the AM system and can be shared when there is a dual AM/FM configuration.

The two most important characteristics of a receiver are:

Sensitivity, which is the minimum signal level that a receiver requires to produce an acceptable output signal-to-noise ratio when the input signal is modulated by a standard amount.

Selectivity, which is the ability of a receiver to reject signals outside a given band while accepting signals that are within that band.

Since the goal of this thesis was to investigate the feasibility and performance of VHDL-AMS, we did not perform the standard RF measurements like gain analysis, noise figure, 1-dB compression point and selectivity and sensitivity analysis. Noise sources were not included in any of the components modeled.

2.5 Related Research

Apart from the research carried out in the Distributed Processing Laboratory at the University of Cincinnati, a similar research in the area of VHDL-AMS modeling is being conducted by the EDA research group, University of Southampton, U.K, Duke University, Analogy Inc. and many others. For more information on the related work in VHDL-AMS modeling and validation refer http://www.vhdl-ams.org.
Chapter 3

VHDL – AMS MODELS

A significant part of this chapter is devoted to the circuit models, their parameters and their modeling in VHDL-AMS at various abstraction levels. We present a few guidelines for the modeling of a VHDL-AMS system. The functional and performance evaluation of these models is presented in the next chapter.

3.1 Guidelines

To ensure the proper execution of the models created in VHDL-AMS, we present a few guidelines that we followed:

1. While modeling a continuous system, the discontinuities in quantities and their derivatives usually encountered while defining the piece-wise behavior of the system, must be defined using a break statement. Identifying where the discontinuities might occur is at times a difficult and uncertain task.

2. VHDL-AMS Models are sensitive to the initial conditions and must be properly established.
In addition to the above guidelines, there are a number of rules in the Language Reference Manual (LRM) that must be followed for the successful execution of the models written in VHDL-AMS such as, at all times the number of quantities (free and through) must equal the number of equations.

### 3.2 Different Abstraction levels

VHDL-AMS allows the description of analog hardware at several levels of abstraction to achieve complete freedom for the designer to define and use models that provide an acceptable balance between accuracy and efficiency. In general, the lowest level of abstraction concerns itself with the implementation issues while the highest-level models what the circuit does, not how it does it. The various abstraction levels that we used and modeled at are discussed below.

**Conceptual Level**

This is the most abstract and highest level in which a system can be defined. In this level, the behavior of the entire system (analog/mixed) is modeled. This approach provides the fastest simulation speed but the least detail in the circuits that are modeled. For example, a simple analog signal amplifier can be modeled with the following simple equation:

\[ \text{Signal\_out} = \text{Signal\_in} \times \text{Amplification}; \]

**Behavioral Level**

The system described in this level of modeling can be a conservative or non-conservative one. In a non-conservative description, the system is described by a set of isolated function that represents input/output relationship (transfer functions). This is also referred to as the signal flow descriptions. The I/O relationships may involve linear and non-linear differential equations. For example, a filter can be defined by its transfer function.

---

In a conservative description, isolated functions are connected through connection points that define two characteristic quantities: the across quantity and the through quantity. These quantities depend upon the nature of the system and they have to be satisfied. For example, a Bipolar Junction Transistor (BJT) can be modeled using KCL/KVL equations.

**Structural Level**

The system is modeled as a connection of subsystems such as resistors, capacitors and transistors. Various components are defined, instantiated and connected as shown in the circuit diagrams in the later part of this chapter. For example: a BJT is modeled by connecting various elements like Resistors, Capacitances, Voltage controlled current source.

One general observation that can be made is that, as we move down the abstraction levels, the complexity and the simulation time increases and also the output is more detailed.

For the behavioral and structural level models of each of the components, please refer appendix A.

### 3.3 COMPONENT MODELING

#### 3.3.1 MIXER

Mixers, or frequency converters, are sometimes called multipliers and in the early days of superheterodyne receivers, they were called "detectors". The mixer has to have following properties: convert the desired signal from the received frequency to the receiver's first IF (intermediate frequency); convert the signal with minimum distortion and/or additive noise; convert the signal with minimum loss; convert the signal with good frequency accuracy.
3.3.1.1 MIXER ACTION

The process by which a mixer operates is the nonlinear processing of a pair of signals in such a way that the sum/or the difference of the two frequencies is produced. Frequency changing is a form of low-level amplitude modulation where in the oscillator acts as the carrier component of the modulating scheme and the RF signal is the modulating signal. The modulation process produces sideband components, which are used as the intermediate frequency. The incoming signal, whether AM or FM, will modulate the resultant intermediate-frequency output of the mixer so that input signal information is preserved but shifted in frequency. This can be accomplished with either diodes or amplifiers, as any device with nonlinear transfer function will work.

The mixer can be of active type or passive type. Active mixers have an active amplifier that provides net conversion gain.

3.3.1.2 PASSIVE MIXER

Passive mixers, which typically use one or more diodes as their nonlinear element, are used primarily in high-quality applications. A typical double balanced mixer [2] is shown in Figure 3.1.

![Double Balanced Mixer Diagram](image)

*Figure 3.1: Double Balanced, diode ring Mixer*
These are called balanced mixers because both of the input signals are suppressed at the output. When the input signal at the RF port swings positive, two diodes in the ring conduct and when the input swings negative the other two diodes conduct. Therefore the input signal at the RF port sees a transformer whose secondary is shorted through a pair of diodes. So does a signal at the LO port. The signals at the IF port are suppressed by balancing the transformer windings at both the input and return connections. Thus all the input signals are shorted through the diodes or are canceled by the transformer. The mutual intermodulation of the current in the diodes generates products of the input signals but does not pass the input signals themselves to the mixer output. Passive mixers are small and have excellent performance but do not lend themselves to large-scale production.

3.3.1.3 ACTIVE MIXER

Active mixers are basically amplifiers that have non-linear transfer functions. That is, an amplifier with or without significant gain that distorts its input signal(s) can be

![A Simple JFET Mixer](image)

Figure 3.2: A Simple JFET Mixer
used as a mixer. Selecting the desired frequency component is accomplished by either tuning the differential transistor collector currents (in case of a differential amplifier configuration) or by placing a filter at the mixer output.

Any amplifier will work as a mixer if it is overdriven or biased at a point at which the signals input to it are distorted. At this point, its transfer function is nonlinear and although it may not be an ideal mixer, it still works as a mixer. A simple JFET (Junction Field Effect Transistor) mixer is shown in Figure 3.2.

The JFET model used is a small signal hybrid PI model. The VHDL-AMS model and the circuit diagram of the JFET is included in the Appendix A and has been omitted here for brevity.

**Mixer Model**

We modeled both the active and passive type of mixer circuit in the conceptual, behavioral and structural levels of abstraction. At the conceptual level, the mixer is modeled simply as the product of the two incoming signals. The VHDL-AMS model of the mixer at the conceptual level is given below:

```vhdl
PACKAGE electricalsystem IS
  NATURE electrical IS real ACROSS real THROUGH;
  FUNCTION SIN(X: real) RETURN real;
  FUNCTION EXP(X: real) RETURN real;
END PACKAGE electricalsystem;

use work.electricalsystem.all;

entity mixer_conceptual is
  port(terminal rf_in, lo_osc, mixer_op, :electrical);
end entity mixer_conceptual;

Architecture mixer_conceptual of mixer_conceptual is
  CONSTANT R_rf: real:=10.0e3;  --> internal res. to sample the voltage
  CONSTANT R_lo: real:=10.0e3;
  quantity v_rf_amplifier across i_rf_amplifier through rf_in to electrical reference;
```
quantity v_local_Oscillator across i_local_Oscillator through lo_osc to electrical reference;
quantity v_mixer across i_mixer through mixer_op to electrical reference;

begin

v_rf_amplifier == i_rf_amplifier * R_rf;
v_local_oscillator == i_local_oscillator * R_lo;
v_mixer == v_rf_amplifier * v_local_oscillator;

end architecture mixer_conceptual_behav;

The passive mixer modeled is a high performance mixer. There is no phase shift introduced by it. The only care that has to be taken is to ensure that the voltage across the diode does not cause it to operate in the breakdown region.

3.3.2 LIMITER

A limiter is a circuit whose output is constant amplitude for all inputs above a critical value. Its function in a FM receiver is to remove any residual amplitude modulation and the amplitude variations due to noise and provide an output that is independent of both input amplitude and rate of change of input amplitude. In addition, the limiter also provides AGC action since signals from the critical minimum value up to the some maximum value provide a constant input level to the discriminator. The

![Figure 3.3: Limiter Input/Output relationship](image)

2 Frequency Modulation
input/output waveforms of a limiter are shown in the following Figure 3.4

There are many types of limiters, such as the overdriven limiter, dynamic limiter, diode limiter and gated-bean tube. We shall only consider the diode limiter due to its ease of construction and its good performance. The circuit diagram of a diode limiter is shown below in the Figure 3.4.

![Limiter Circuit Diagram](image)

**Figure 3.4 : Limiter circuit diagram**

**Limiter Model**

The Mixer is separately modeled in the conceptual, behavioral and structural abstraction levels. Since the function of the limiter is to limit the peak-to-peak voltage of the input, the conceptual model can be described in VHDL-AMS as follows:

```vhdl
PACKAGE electricalsystem IS
  nature electrical IS real across real through;
  FUNCTION SIN (X : real ) RETURN real;
  FUNCTION exp  (X : real ) RETURN real;
END PACKAGE electricalSystem;

use work.electricalsystem.all;
entity limiter_conceptual is
  generic(V_limit:real:=1.5);
  port(terminal Signal_in,Signal_out: electrical);
```
end entity limiter_conceptual;

architecture lim_conceptual of limiter_conceptual is
  quantity v_Signal_in across i_Signal_in through Signal_in to electrical'reference;
  quantity v_Signal_out across i_Signal_out through Signal_out to electrical'reference;
begin
  v_Signal_in == i_Signal_in *10.0e4;
  if (v_Signal_in>=0.0 and v_Signal_in >= v_limit) use
    v_signal_out == v_limit;
  elsif (v_Signal_in>=0.0 and v_Signal_in < v_limit) use
    v_Signal_out == v_Signal_in;
  elsif (v_Signal_in<0.0 and v_Signal_in < -v_limit) use
    v_Signal_out == -v_limit;
  elsif (v_Signal_in<0.0 and v_Signal_in >= -v_limit) use
    v_Signal_out == v_Signal_in;
  end use;
end architecture lim_conceptual;

The limiter introduces no phase shift and operates correctly as long as the voltage across the diodes doesn’t exceed the breakdown voltage.

3.3.3 FM Discriminator (Demodulator)

Detection or demodulation is defined as the process of recovering the intelligence from a high-frequency wave. The input of the FM detector comes from the limiter and the output of the FM detector goes to the AF amplifier. No part of the receiver is more critical to its performance than its demodulator. Any nonlinear circuit element can be used to restore the intelligence while the load circuit, acting as a low-pass filter, removes the high frequency component from the demodulated output.

There are different kinds of FM detectors (discriminators) like the slope detector, ratio detector, pulse count discriminator, Phase Lock demodulator and the Quadrature detector. In this thesis, we shall discuss the working of the slope discriminators and their modeling.
The Slope detector followed Foster-Seeley and ratio demodulators and solved some of their problems with its sheer simplicity [3]. A simple slope detector actually works by converting the frequency modulation to amplitude modulation and then AM demodulating the signal. It does this by passing the FM signal through an amplifier or bandpass filter that is deliberately detuned with respect to the input FM carrier. For example, if the intermediate frequency modulated signal is the typical 10.7 MHz, then the RF stage is tuned at least 100 kHz above or below the 10.7 MHz signal. Thus, the signal falls on the slope of the amplifier’s response curve, causing the signals at one frequency to be amplified a different amount than signals at another frequency and converting the modulation from FM to AM. The signal can then be demodulated with the envelope detector or with careful design, in the same amplifier used to convert from FM to AM.

The Travis detector is shown in Figure 3.5. The primary may or may not be tuned. The primary circuit is broadly tuned to accept the full carrier swing. Ls1 and Ls2 are inductively coupled to Lp but there is essential no coupling between Ls1 and Ls2. Ls1-Cs1 resonates at $f_1$ and Ls2-Cs2 resonates at $f_2$, the center frequency (10.7 MHz) appearing midway between both. The composite characteristic curve is shown in the Figure 3.6. The secondary can be considered as two simple slope detectors connected in parallel. The detection process that takes place here is the same type of translation that is described in the above paragraph. The discharge time constant of both R1-C1 and R2-C2 is 10 microseconds, which is long compared to the IF. Therefore, it follows the envelope of the curve and results in the audio detection. The circuit parameters are listed in the Table 3.1.

![Figure 3.5: Travis detector (Slope Detector) circuit](image-url)
Table 3.1: circuit parameters of the travis detector

<table>
<thead>
<tr>
<th>Component</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_p$</td>
<td>0.2212 pF</td>
</tr>
<tr>
<td>$L_p$</td>
<td>1.0 mH</td>
</tr>
<tr>
<td>$L_s1$</td>
<td>0.5 mH</td>
</tr>
<tr>
<td>$L_s2$</td>
<td>0.5 mH</td>
</tr>
<tr>
<td>$C_s1$</td>
<td>0.2181 pF</td>
</tr>
<tr>
<td>$C_s2$</td>
<td>0.2246 pF</td>
</tr>
<tr>
<td>$C_1$</td>
<td>100.0 pF</td>
</tr>
<tr>
<td>$C_2$</td>
<td>100.0 pF</td>
</tr>
<tr>
<td>$R_1$</td>
<td>100.0 kΩ</td>
</tr>
<tr>
<td>$R_2$</td>
<td>100.0 kΩ</td>
</tr>
<tr>
<td>$I_{sat(diode)}$</td>
<td>1.0e$^{-14}$</td>
</tr>
<tr>
<td>$N(diode)$</td>
<td>1.0</td>
</tr>
</tbody>
</table>

**Figure 3.6**: Output Characteristics of the travis detector
**Travis Detector Model**

It is not possible to define a conceptual level model of the detector. Hence, it has been modeled only in the behavioral and structural levels whose VHDL-AMS models are given in the appendix.

The Slope detectors tend to be more stable and they will continue to work as long as the signal falls on the amplifier’s tuning slope. They are, in general however, less accurate than many FM demodulators because of the imperfect linearity in a typical tuning slope.

3.3.4 FILTER

Filters are circuits that are capable of passing input signals with certain selected frequencies through to the output while rejecting signals with other frequencies. Filters use active devices such as transistors or op-amps and passive RC networks. We have modeled both the active and passive filters of all kinds (low-pass, high-pass, band-pass and band-stop). We have used the ideal op-amp model in our active filters. All the filters modeled have butterworth characteristics. The circuit diagrams of the passive high pass

![Figure 3.7.1: A Passive Low Pass Filter](image)

![Figure 3.7.2: A Passive High Pass Filter](image)
and low pass filters are shown in the Figure 3.7.2 and 3.7.1 respectively. The value of C is a generic in the all the models and is computed for every critical frequency ($f_c$).

The circuit diagrams for the active low and high pass filters is shown in the Figure 3.8. We have only shown the low and high pass filters. Band pass and band-stop filters are modeled using the low and high pass filters as discussed in the appendix A.

![Low Pass Filter Using an OP-AMP](image1)

*Figure 3.8.1: Low Pass Filter Using an OP-AMP*

![High Pass Filter Using an OP-AMP](image2)

*Figure 3.8.2: High Pass Filter Using an OP-AMP*
For obtaining the *butterworth* response, the values of R1 and R2 are fixed at 586.0 Ohms and 1000.0 Ohms. The values of R_a and R_b are fixed at 1000.0 Ohms each and the value of the capacitances are computed for the critical value supplied (as the generic value). This enables the circuit to work over a wide range of frequencies. The filters modeled using the ideal op-amp are highly selective and the circuit operates correctly as long as the input voltages do not exceed the saturation voltage of the op-amp

**Filter Model**

The active and passive (low and high pass) filters are modeled in the behavioral and structural level and also using the transfer functions. The VHDL-AMS models of both kinds of filters using *ltf* [12] is given below:

```vhdl
-- LTF model of a 2nd Order Low Pass Filter
PACKAGE electricalsystem IS
    NATURE electrical IS real ACROSS real THROUGH;
    TYPE real_vector is array(natural range<> ) of real;
    FUNCTION SIN(X:real) RETURN real;
    FUNCTION EXP(X:real) RETURN real;
END PACKAGE electricalsystem;

------------------------- LPF -------------------------
use work.electricalsystem.all;

entity low_pass is
end entity low_pass;

architecture basic of low_pass is
    terminal T1,T2 :electrical;
    CONSTANT Wc : real:=2.0*3.141592654*108.0e6;
    CONSTANT den: real_vector(2 downto 0):=(1.0, 1.412*Wc,1.0*Wc*Wc);
    CONSTANT num: real_vector(1 downto 0):=(0.0, 1.0*Wc*Wc);
    quantity v_ip  across i_ip  through t1;
    quantity v_out across i_out through T2;
BEGIN
    vsource :V_ip ==(1.0*sin((2.0*22.0/7.0*108.0e6)*real(time'pos(now))*1.0e-15));
    v_out ==V_ip'ltf(num,den);
```

end architecture basic;

-- LTF model of a 2nd Order HIGH Pass Filter

PACKAGE electricalsystem IS
  NATURE electrical IS real ACROSS real THROUGH;
  TYPE real_vector is array(natural range<> ) of real;
  FUNCTION SIN(X:real) RETURN real;
  FUNCTION EXP(X:real) RETURN real;
END PACKAGE electricalsystem;

------------------------- HPF -------------------------
use work.electricalsystem.all;

entity high_pass is
end entity high_pass;

architecture basic of high_pass is
  end entity high_pass;

  terminal T1,T2 :electrical;
  
  CONSTANT Wc : real:=2.0*3.141592654*88.0e6;
  -- |
  -- --> cutoff frequency
  CONSTANT den: real_vector(2 downto 0):=(1.0, 1.412*Wc,1.0*Wc*Wc);
  CONSTANT num: real_vector(2 downto 0):=(1.0, 0.0, 0.0);

  quantity v_ip across i_ip through t1;
  quantity v_out across i_out through T2;

  BEGIN
    vsource :V_ip ==(1.0*sin((2.0*22.0/7.0*10.0e6)*real(time'pos(now))*1.0e-15));
    v_out ==V_ip'ltf(num.den);

end architecture basic;
3.3.5 Voltage controlled Oscillator

The Voltage Controlled Oscillator (VCO) is an oscillator whose frequency can be changed by a variable dc control voltage. In this thesis, we have implemented only the conceptual level VHDL-AMS model of the VCO. Further research can be done in this and models developed for a wide range of oscillators. The conceptual level VHDL-AMS model of the VCO is given below:

```vhdl
PACKAGE electricalsystem IS
  NATURE electrical IS real ACROSS real THROUGH;
  FUNCTION SIN(X:real) RETURN real;
  FUNCTION EXP(X:real) RETURN real;
END PACKAGE electricalsystem;

use work.electricalsystem.all;

entity VCO is
  generic(
    amp: real := 1.0;    -- Amplitude of the VCO
    fc : real := 1.0E6;  -- VCO frequency at Vc
    df : real := 75.0e3; -- [Hz/V], frequency characteristic slope
    Vc : real := 0.0     -- center frequency input voltage
  );
  port(terminal vol_in, vco_op: electrical);
end entity VCO;

architecture conceptual of VCO is

  CONSTANT TWO_PI: real := 6.28318530718; -- 2pi
  quantity v_in across i_in through vol_in to electrical'reference;
  quantity v_op across i_op through vco_op to electrical'reference;

begin

  break v_in => Vc;
  V_op == (amp*sin(TWO_PI*(fc+(df*(v_in-Vc))))*real(time'pos(now))*1.0e-15));

end architecture conceptual;
```
3.3.6 Amplifier

In this section, we present the models of the different types of amplifiers [14] that were used as a part of this research. The amplifiers modeled are basically classified into two types: untuned amplifiers and tuned amplifiers. The untuned amplifiers provide a signal gain that is non-dependent on the frequency of operation while the tuned amplifiers amplify a selected band of frequencies, rejecting the others. Any bandpass filter can work as a tuned amplifier. To be concise, this thesis presents only the circuits used along with their circuit parameters, leaving out the theory of operation of the amplifier.

The transistor model used in the amplifier circuit is a small signal model. The BJT is a current-controlled device, where base current modulates the collector current of the transistor. The small signal equivalent circuit of the silicon bipolar transistor has been derived and its equivalent circuit along with its various parameters [5] is presented below in Figure 3.9.

![Trans T-equivalent Model of the HF transistor](image)

**Figure 3.9**: Trans T-equivalent Model of the HF transistor
The untuned amplifier is shown along with its circuit parameters in Figure 3.10.

Figure 3.10: Two stage Untuned Amplifier
The BJT parameters are supplied earlier in the earlier Figure. The tuned amplifier along with its parameters is shown in the Figure 3.11. For the tuned amplifier, the ‘Q’ value of the tuning element has to be properly set for the proper selectivity of the amplifier.

A Wein-bridge oscillator and an op-amp (using the small signal transistor model) have also been modeled and its VHDL-AMS code included in Appendix A).
Chapter 4

Performance Evaluation

This chapter presents the results of simulating the models investigated and described in the previous chapter. Our purpose is to demonstrate that VHDL-AMS is indeed capable for describing and simulating high frequency circuits in the time domain. We evaluate the performance of the VHDL-AMS models as compared to those in SPICE\(^1\), by incorporating them in several test cases and exercising them in there respectively simulators. We discuss the criteria used to evaluate and then proceed to compare the results of SEAMS\(^2\) and SPICE based on the criteria.

4.1 Criteria

4.1.1 Error Criteria

The criteria used in this thesis for the comparison of the two waveforms is the error between them [6][7]. There are two kinds of error

- **Absolute Error** The error between the measured waveform and the reference waveform
- **Relative Error** The ratio of the absolute error to the reference waveform.

\(^1\) SPICE is considered as the benchmark to validate the models
To compute the absolute error between the two waveforms with asynchronous sampling, we first interpolate one waveform with the other and calculate the sample-by-sample difference. The relative error (in dB) is then computed from the absolute error as follows

\[
\text{Relative Error} = 20.0 \log \left( \frac{\text{Maximum error}}{\text{Peak-Peak value of reference waveform}} \right)
\]

We consider the conceptual model of a RF component to be of acceptable fidelity and quality if the error between VHDL-AMS simulation and the SPICE simulation \[11\] of the model is less than –20.0 dB. For the behavioral level models, the error has to be less than –25.0 dB for it to be acceptable. The error has to be less than –30.0 dB for a structural level model.

### 4.1.2 Execution Time Criteria

The execution time of the models is utilized in the feasibility study. The total simulation time of an VHDL-AMS model is comprised of three parts \[8\]:

- Time taken by scram (parsing and syntax checking) to produce the C++ code.
- Time taken by g++ compiler to create the object files.
- Time taken by the seams simulator.

We use only the time taken by seams in our comparison study because this is the actual time taken for the simulation of the model. The time taken by scram and g++ compiler is the time it takes to produce the executable code for the simulator.

For the same model, SPICE and SEAMS produce waveform points at different times. Hence, we formulate a measure for comparing the performance of the simulators based on the average execution time per solution point that is determined as follows:

---

2 Simulation Environment for VHDL-AMS
Average execution time per point (Ave. time) = \frac{\text{Total Execution Time}}{\text{Total points solved}}

4.2 Simulation Environment

All models were executed on the same processor. The specific characteristics of this are:

<table>
<thead>
<tr>
<th>Simulation Platform</th>
<th>Sun4u</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating System</td>
<td>Sun Solaris 5.7</td>
</tr>
<tr>
<td>Processor Type</td>
<td>Sparc</td>
</tr>
<tr>
<td>Number of Processors</td>
<td>4</td>
</tr>
</tbody>
</table>

SEAMS version 1.1C was used in the research and the C++ compiler used to run SEAMS was gcc 2.7.2.1. Spice3f4 was used to run the spice models.

4.3 Performance Evaluation

We present the performance of each RF component by exercising them using suitable test benches. The waveforms obtained by exercising these test-circuits (i.e., testbench together with the circuit under test) in the simulators (SPICE and SEAMS) are compared and the errors computed.

The FM receiver could not be simulated as a single model. To obtain even a few cycles of the output waveform (e.g. 10 cycles for $F_{\text{signal}} = 10.0$ kHz), we had to simulate the system for 1.0 ms during which time $10^5$ cycles of the FM carrier signal (e.g. $F_{\text{carrier}} = 100.0$ MHz) occur. Simulating this number of cycles on SEAMS is predicted to take more than seven days! Thus we split the FM receiver into two parts as shown in Figure 4.2 and simulated each one independently. There is no direct link between the

\* Only a single processor participates in the execution of both SEAMS and SPICE.
output of the limiter and input of the discriminator. The reason for the splitting of the FM receiver at the output of the limiter is due to the large frequency difference that exist between the ends of the discriminator. The first part is simulated using the testbench described later in section 4.3.7 on page 61 and assuming its correct output, an analytical model of it (output waveform of the first part) is used as the source (ISI) for simulating the second part. The frequency modulation in the output of the first part was not observable, as it was quite small when compared to the Intermediate Frequency (75.0 kHz modulation on an intermediate frequency of 10.7 MHz). Hence the correct operation of the first part is not guaranteed. By simulating the first part in the frequency domain, the required data can be obtained and its operation can be guaranteed.

For each of the components, we present the test circuit and its output and then proceed to compute the actual and relative errors. The average time for each component is shown the Tables 4.1 and 4.2 on pages 64 and 65 respectively.

**4.3.1 LIMITER**

A limiter is basically a double clipper. The test bench for the limiter is as shown in Figure 4.1. The input signal (for demo purposes only) is a pure sine wave; no bias,
noise or modulation is applied. The circuit diagram is shown in Figure 3.4 and the VHDL-AMS code for the limiter is given in section A.2. The diode models used were developed at the University of Cincinnati [10].

The limiter is separately modeled at three abstraction levels (structural, behavioral and conceptual).

**Structural Model**

The output waveform from the VHDL-AMS and SPICE model is superimposed and shown below in Figure 4.3.

*Figure 4.2: Testbench for the limiter circuit*

*Figure 4.3: Output from the structural model of the limiter. The VHDL-AMS and the SPICE waveforms are in close agreement and overlap each other*
It can be seen that the two waveforms are in close agreement. The absolute error between the waveforms is computed and shown in the Figure 4.4. As observed, the maximum error is no more than 0.15 V for a peak-to-peak voltage of 7.60 V. The relative error is $20 \times \log \left(\frac{0.15}{7.60}\right) = -34.09$ dB.

![Absolute Error of the structural model of the limiter](image)

Figure 4.4: Absolute Error of the structural model of the limiter

The execution time and other related data for the VHDL-AMS model is tabulated in Table 4.1 on page 64 and the data for the SPICE model is tabulated in Table 4.2 on page 65.

**Behavioral Level**

The output waveform of the VHDL-AMS and SPICE model is shown in Figure 4.5. The absolute error between the waveforms is computed and shown in the Figure 4.6. It can be seen that the waveforms are in close agreement. As observed the maximum error is no more than 0.18 V for a peak-to-peak voltage of 7.60 V. The relative error is $-32.51$ dB.
The execution time and other related data for the VHDL-AMS model is tabulated in Table 4.1 on page 64 and the related data for the SPICE model is tabulated in Table 4.2 on page 65.

Figure 4.5: Output from the behavioral model of the limiter. The two waveforms overlap each other.

Figure 4.6: Absolute Error for the behavioral model of the Limiter
Conceptual Level

The output waveform of the VHDL-AMS and SPICE model is shown in Figure 4.7.

Figure 4.7: Output from the conceptual model of the Limiter

Figure 4.8: Absolute Error for the conceptual model of the Limiter
The absolute error between the waveforms is computed and shown in the Figure 4.8. It can be seen that the waveforms are in close agreement. As observed, the maximum error is no more than 0.13 V for a peak-to-peak voltage of 7.60 V. The relative error is –35.33 dB. The execution time and other related data for the VHDL-AMS model is tabulated in Table 4.1 on page 64 and the related data for the SPICE model is tabulated in Table 4.2 on page 65.

The output characteristics of the limiter do not depend on the operating frequency. It can be observed from the above plots that the limiter exhibits good fidelity and performance. Time domain modeling is found to be effective for modeling limiters.

<table>
<thead>
<tr>
<th>Model</th>
<th>Peak-to-Peak Voltage</th>
<th>Maximum Error in Volts</th>
<th>Error in dB</th>
<th>Comparison with the specifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conceptual</td>
<td>7.60</td>
<td>0.19</td>
<td>-32.04</td>
<td>-12.04 dB</td>
</tr>
<tr>
<td>Behavioral</td>
<td>7.60</td>
<td>0.18</td>
<td>-32.51</td>
<td>-7.51 dB</td>
</tr>
<tr>
<td>Structural</td>
<td>7.60</td>
<td>0.15</td>
<td>-34.09</td>
<td>-4.09 dB</td>
</tr>
</tbody>
</table>

*Table 4.3: Performance of the various limiter models*

### 4.3.2 MIXER

The testbench used for the mixer circuit is shown below in Figure 4.9.

*Figure: 4.9: Testbench for the mixer circuit*
The circuit diagram is shown in Figure 3.1 and the VHDL-AMS code for the mixer is given in section A.3. The mixer has been separately modeled at three abstraction levels.

**Structural Level**

The output waveform from the VHDL-AMS and SPICE model is superimposed and shown below in Figure 4.10.

![Figure 4.10: Output from the structural model of the Mixer. The VHDL-AMS and the SPICE waveforms overlap each other.](image)

![Figure 4.11: Absolute Error for the structural model of the Mixer](image)
It can be seen that the waveforms are in close agreement. The absolute error between the waveforms is computed and shown in the Figure 4.11. As observed, the maximum error is no more than 1.2 mV for a peak-to-peak voltage of 21.3 mV. The relative error is –24.98 dB. The execution time and other related data for the VHDL-AMS model is tabulated in Table 4.1 on page 64 and the related data for the SPICE model is tabulated in Table 4.2 on page 65.

**Behavioral Level**

The output waveform from the VHDL-AMS and SPICE model is superimposed and shown below in Figure 4.12.

![Figure 4.12: Output from the behavioral model of the Mixer](image)

**Figure 4.12: Output from the behavioral model of the Mixer**

It can be seen that the waveforms are in close agreement. The absolute error between the waveforms is computed and shown in the Figure 4.13.
As observed, the maximum error is no more than 1.0 mV for a peak-to-peak voltage of 21.53 mV. The relative error is –26.56 dB.

The execution time and other related data for the VHDL-AMS model is tabulated in Table 4.1 on page 64 and the related data for the SPICE model is tabulated in Table 4.2 on page 65.

**Conceptual Level**

The output waveform from the VHDL-AMS and SPICE models is superimposed and shown below in Figure 4.14. It can be seen that the waveforms are in close agreement. The absolute error between the waveforms is computed and shown in the Figure 4.15.

As observed, the maximum error is no more than 22.0 µV for a peak-to-peak voltage of 76.0 µV. The relative error is computed and is –10.76 dB.

The execution time and other related data for the VHDL-AMS model is tabulated in Table 4.1 and the related data for the SPICE model is tabulated in Table 4.2.
Figure 4.14: Output from the conceptual model of the Mixer

Figure 4.15: Absolute Error for the conceptual model of the Mixer
From the above plots it can be concluded that the mixer modeled exhibits good fidelity and performance. The maximum error is -10.76 dB for the conceptual level model. The gain of the mixer is very low and a high-gain IF amplifier should follow it. An active mixer has also been modeled and simulated and it also exhibits good performance. Its code has been included in the appendix A and has been left here for brevity. The above mixer can be effectively used as a discriminator.

The sum and difference frequencies, harmonics, and the original signal input frequencies are usually present in the mixer output. While modeling a mixer, we are primarily interested in its output spectra. This is not possible to obtain in the time domain. Frequency domain modeling and simulation provides such data and hence mixers can be more effectively modeled in the frequency domain.

<table>
<thead>
<tr>
<th>Model</th>
<th>Peak-to-Peak Voltage in V</th>
<th>Maximum Error in Volts</th>
<th>Error in dB</th>
<th>Comparison with the specifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conceptual</td>
<td>76.00 µV</td>
<td>22.0 µV</td>
<td>-10.76</td>
<td>10.24 dB</td>
</tr>
<tr>
<td>Behavioral</td>
<td>21.3 mV</td>
<td>1.0 mV</td>
<td>-24.98</td>
<td>0.02 dB</td>
</tr>
<tr>
<td>Structural</td>
<td>21.53 mV</td>
<td>1.2 mV</td>
<td>-26.56</td>
<td>3.44 dB</td>
</tr>
</tbody>
</table>

*Table 4.4 : Performance of the various mixer models*

4.3.3 RF AMPLIFIER

The test bench for the RF Amplifier is shown below. The input signal (for demo purposes only) is a pure sine wave; no bias, noise or modulation is applied. The circuit diagram is shown in Figure 3.10 and the VHDL-AMS code is given in section A.4. The RF Amplifier is separately modeled at three abstraction levels (structural, behavioral and conceptual). The output waveforms and error curves for the behavioral and structural models are the same and are shown only once.
Structural/Behavioral Level

The output waveform from the VHDL-AMS and SPICE model is superimposed and shown below in Figure 4.17.

The absolute error between the waveforms is computed and shown in the Figure 4.18.
As observed from the above waveform the absolute error is no more than 0.00009 V for a peak-to-peak voltage of 0.0076 V. The relative error is $20 \times \log\left(\frac{0.00009}{0.0076}\right) = -38.97$ dB.

The execution time and other related data for the VHDL-AMS model is tabulated in Table 4.1 on page 64 and the related data for the SPICE model is tabulated in Table 4.2 on page 65.

*Conceptual Level*

The output waveforms from the VHDL-AMS and SPICE models is superimposed and shown below in Figure 4.19.
It can be seen that the two waveforms are in close agreement. The absolute error is computed and plotted in the Figure 4.20. As observed, the absolute error is no more than $190.0 \, \mu V$ for a peak-to-peak voltage of 0.0057 V. The relative error is $-29.84$ dB.

![Figure 4.19: Output from the conceptual model of the RF Amplifier](image1)

![Figure 4.20: Absolute Error for the conceptual model of the RF Amplifier](image2)
The execution time and other related data for the VHDL-AMS and SPICE model are tabulated in Table 4.1 and Table 4.2 respectively.

The gain that is obtained from the two-stage amplifier is 49.25dB. For higher gains, the number of amplifier stages needs to be suitably increased. While modeling a RF amplifier, we are mainly interested in its gain and transient response. These characteristics are easily obtained in time domain and it is better to stick with the time domain while modeling RF untuned amplifiers.

<table>
<thead>
<tr>
<th>Model</th>
<th>Peak-to-Peak Voltage in V</th>
<th>Maximum Error in Volts</th>
<th>Error in dB</th>
<th>Comparison with the specifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conceptual</td>
<td>0.000457</td>
<td>0.00019</td>
<td>-29.84</td>
<td>-9.84 dB</td>
</tr>
<tr>
<td>Behavioral</td>
<td>0.0076</td>
<td>0.000009</td>
<td>-38.97</td>
<td>-13.97 dB</td>
</tr>
<tr>
<td>Structural</td>
<td>0.0076</td>
<td>0.000009</td>
<td>-38.97</td>
<td>-8.97 dB</td>
</tr>
</tbody>
</table>

*Table 4.5: Performance of the various RF Amplifier models*

### 4.3.4 FILTERS

Active filters are modeled [9] and used extensively in this thesis. The three different types of filters (low-pass, high-pass, band-pass) are separately modeled. We have used an ideal op-amp in all these filters. The output waveforms and error plots of structural and behavioral level models are the same and are treated only once. Each filter type is dealt separately.
The test bench for the filter is shown below in Figure 4.21. The input signal (for demo purposes only) is a pure sine wave; no bias, noise or modulation is applied. The circuit diagram is shown in Figure 3.8 and the VHDL-AMS code is given in section A.7. Each kind of filter is simulated for a wide range of input signals and its output characteristic is plotted.

**LOW PASS FILTER**

The test circuit of a low-pass filter is shown in Figure 4.21. The output waveforms from SEAMS and SPICE overlap.

![Figure: 4.21 : Testbench for the filter circuit](image)

*Figure 4.22 : Output from the struct/behav model of the LPF. The output waveforms from SEAMS and SPICE overlap.*
waveforms from the VHDL-AMS and SPICE models for a particular frequency is superimposed and shown below in Figure 4.22. It can be seen that the two waveforms are in close agreement and the absolute error plot is shown in Figure 4.23.

![Absolute Error for the struct/behav model of the LPF](image)

As observed, the maximum absolute error is no more than 0.021V for a peak-to-peak reference of 2.42 V. The relative error is

\[ 20 \times \log \left( \frac{0.021}{2.42} \right) = -41.23 \text{ dB} \]

The execution time and other related data for the VHDL-AMS and SPICE model are tabulated in Table 4.1 and Table 4.2 respectively.

The frequency response of the LPF is shown below in Figure 4.24.
HIGH PASS FILTER

The test circuit of a high-pass filter is shown in Figure 4.21. The output waveforms from SEAMS and SPICE overlap.
waveforms from the VHDL-AMS and SPICE models for a particular frequency is superimposed and shown below in Figure 4.25. It can be seen that the two waveforms are in close agreement and the absolute error plot is shown in Figure 4.26.

![Figure 4.26: Absolute Error for the struct/behav model of the HPF](image)

As observed, the maximum absolute error is no more than 0.018 V for a peak-to-peak reference voltage 1.521 V. The relative error is 

\[20 \times \log \left(\frac{0.024}{3.042}\right) = -42.05 \text{ dB}\]

The execution time and other related data for the VHDL-AMS and SPICE model are tabulated in Table 4.1 and Table 4.2 respectively.

The frequency response of the HPF is shown below in Figure 4.27.
**BAND PASS FILTER**

The test circuit of a band-pass filter is shown in Figure 4.21. The output waveforms from SEAMS and SPICE overlap.

![Frequency Response of a HPF](image)

*Figure 4.27: Frequency Response of a High Pass Filter*

![Output from the struct/behav model of the BPF. The output waveforms from SEAMS and SPICE overlap.](image)
waveforms from the VHDL-AMS and SPICE models for a particular frequency is superimposed and shown below in Figure 4.28. It can be seen that the two waveforms are in close agreement and the absolute error plot is shown in Figure 4.29.

![Figure 4.29: Absolute Error for the struct/behav model of the BPF](image)

As observed, the maximum error is no more than 0.0215 V for a peak-to-peak reference voltage of 2.982 V. The relative error is $20 \times \log \left( \frac{0.0215}{2.982} \right)$

$$= -42.84 \text{ dB}$$

The execution time and other related data for the VHDL-AMS and SPICE model are tabulated in Table 4.1 and Table 4.2 respectively.

Form the above plots it can be concluded that the filters exhibit good performance and have good fidelity. The maximum error for various filters is as follows:

- Low Pass Filter $-41.23$ dB
- High Pass Filter $-42.05$ dB
- Band Pass Filter $-42.84$ dB

The frequency response of the BPF is shown below in Figure 4.30.
The filters provide a –40.0 dB roll-off and using these as the building blocks, higher order filters can be constructed. Active filters using a simple op-amp (having 5 BJT\(^1\)s) has also been modeled and they too exhibit similar results. But they are too slow for simulation and hence are not included in our final FM system. Filters have also been modeled using \(\text{ltf's}\). Their output characteristics are similar are left out for brevity. Its error values are shown in Table 4.6. While modeling filters we are mainly interested in observing its frequency selectivity. This is not simple to obtain in the time domain. Frequency domain provides for such data and filters can be more effectively modeled in the frequency domain inorder to study its overall characteristics.

4.2.5 FM DETECTOR

The FM detector that has been modeled and used in our research is the travis discriminator. The testbench is as shown in Figure 4.31.

---

1 Bipolar Junction Transistor
The circuit diagram is shown in Figure 3.3.3 and the VHDL-AMS code for the detector is given in section A.6. The Travis detector has been modeled in the behavioral and structural levels. The output waveforms and the error curve of each are the same.

**Behavioral/Structural Levels**

The output waveforms from the VHDL-AMS and SPICE models is superimposed and shown below in Figure 4.32. It can be seen that the two waveforms are in close agreement. The absolute computed error is plotted in the Figure 4.33.

*Figure 4.31: Testbench for the Travis detector*

*Figure 4.32: Output from the struct/behav model of the FM Detector. The waveforms from SEAMS and SPICE overlap each other.*
As observed, the absolute error is no more than 0.0031 V for a peak-to-peak voltage of 0.023 V. The relative error is $20 \times \log \left( \frac{0.0031}{0.023} \right)$

$$= -17.41 \text{ dB}$$

The large error is mainly due to the interpolation technique used in the error calculation. The output is not a smooth curve though it appears so in the above graph. The execution time and other related data for the VHDL-AMS and SPICE model are tabulated in Table 4.1 and Table 4.2 respectively.

FM detectors extract a low frequency signal from a high frequency modulated signal. To get a reasonable number of cycles of output waveform, we have to simulate the model for a long time and the time domain modeling does not lend itself to harmonic studies. Frequency domain modeling of the FM discriminator can effectively capture its overall characteristics and can be used for performance improvements.

Figure 4.33: Output from the struct/behav model of the FM Detector
4.3.6 IF AMPLIFIER

The test bench for an IF amplifier is similar to that of a RF amplifier and the circuit parameters are kept the same except for the amplitude of the input signal which is maintained at 15.0 mV. The IF amplifier amplifies signals that fall within its bandwidth and rejects all other signals. The IF amplifier has been simulated for various frequencies and its output Vs frequency characteristic is computed. The IF amplifier has been separately modeled at two abstraction levels. The gain of the IF amplifier is 38.5 dB. The output characteristics for the IF amplifier is shown below is Figure 4.35.

![Output characteristics of IF Amplifier](image)

*Figure 4.35: Frequency response of the IF Amplifier*

The execution time and other related data for the VHDL-AMS and SPICE model are tabulated in Table 4.1 and Table 4.2 on pages.

A phase difference was observed (due to the primitive high frequency transistor model used) and this lead to poor fidelity and slightly high error. A bandpass filter can also be used as an IF amplifier. While dealing with IF amplifiers we are interested in its selective amplification and it can only be observed in the frequency domain. The bandwidth of an IF amplifier can be easily obtained in the frequency domain and the circuit can easily fine tuned to obtain greater performance. Hence frequency domain is the ideal domain for modeling IF amplifiers.
4.3.7 FM Receiver

The first part of the FM receiver as described in the beginning of the chapter is constructed from its various components. The testbench for the system is shown below in Figure 4.36.

The abstraction level models for the various components involved are as follows:

- RF Amplifier: Structural Model
- Band Pass Filter: Structural Model
- Mixer: Structural Model
- IF Amplifier: Structural Model
- Limiter: Structural Model

The output is shown below in Figure 4.37. This system was not validated with SPICE. By studying the output, it can be safely stated that the system operates as per our expectations. The frequency modulation of the intermediate frequency cannot be noticed, as it is very small. The frequency spectrum of the output is needed to guarantee its proper functioning but VHDL-AMS does not provide frequency domain modeling and simulation capabilities.
The second part of the FM receiver consists of the Travis detector and the audio amplifier. The testbench for the second part is shown below in Figure 4.38. Assuming correct operation of the first part, an analytical model of a source was used to simulate the interface signal for the second part.

The abstraction level models for the various components involved are as follows

- Travis detector: Structural Model
- Audio Amplifier: Structural Model
The output is shown in Figure 4.39. It can be seen that the second part efficiently extracts the modulating signal from the Intermediate Frequency modulated waveform.

![Output from second part of the receiver](image)

*Figure 4.39: Output from the part 2 of the FM receiver*

From all the results presented above, it can be safely stated that VHDL-AMS is indeed capable of describing high frequency systems. Certain features (described in Chapter 2) need to be incorporated into VHDL-AMS to further increase its applicability. To summarize, the relative error for various abstraction level models and the ratio of execution time of SEAMS Vs SPICE is provided in Table 4.6 on page 66.
### Execution time of VHDL-AMS models

<table>
<thead>
<tr>
<th>Model</th>
<th>Abstraction level</th>
<th>Time (in sec)</th>
<th>Analog Points</th>
<th>Iterations</th>
<th>Rollbacks</th>
<th>Step Reductions</th>
<th>Avg. Exe. Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>scram</td>
<td>g++</td>
<td>seams</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Limiter</td>
<td>Conceptual</td>
<td>0.996</td>
<td>120.214</td>
<td>0.561</td>
<td>203</td>
<td>410</td>
<td>7.765</td>
</tr>
<tr>
<td></td>
<td>Behavioral</td>
<td>1.076</td>
<td>120.24</td>
<td>1.447</td>
<td>203</td>
<td>1037</td>
<td>7.132</td>
</tr>
<tr>
<td></td>
<td>Structural</td>
<td>1.422</td>
<td>228.042</td>
<td>2.616</td>
<td>221</td>
<td>1223</td>
<td>11.845</td>
</tr>
<tr>
<td>Mixer</td>
<td>Conceptual</td>
<td>1.54</td>
<td>274.89</td>
<td>1.159</td>
<td>303</td>
<td>718</td>
<td>3.825</td>
</tr>
<tr>
<td></td>
<td>Behavioral</td>
<td>2.18</td>
<td>246.51</td>
<td>20.539</td>
<td>1152</td>
<td>3469</td>
<td>17.828</td>
</tr>
<tr>
<td></td>
<td>Structural</td>
<td>2.09</td>
<td>297.33</td>
<td>28.716</td>
<td>1138</td>
<td>2295</td>
<td>25.234</td>
</tr>
<tr>
<td>Mixer</td>
<td>Conceptual</td>
<td>0.71</td>
<td>47.39</td>
<td>1.8065</td>
<td>203</td>
<td>609</td>
<td>8.899</td>
</tr>
<tr>
<td></td>
<td>Behavioral</td>
<td>1.15</td>
<td>120.25</td>
<td>1.631</td>
<td>220</td>
<td>420</td>
<td>7.33</td>
</tr>
<tr>
<td></td>
<td>Structural</td>
<td>1.602</td>
<td>265.9</td>
<td>1.661</td>
<td>220</td>
<td>444</td>
<td>7.55</td>
</tr>
<tr>
<td>Low Pass Filter</td>
<td>Conceptual</td>
<td>0.65</td>
<td>47.59</td>
<td>1.02</td>
<td>203</td>
<td>613</td>
<td>6.024</td>
</tr>
<tr>
<td></td>
<td>Behavioral</td>
<td>1.17</td>
<td>119.95</td>
<td>1.604</td>
<td>207</td>
<td>418</td>
<td>7.753</td>
</tr>
<tr>
<td></td>
<td>Structural</td>
<td>1.626</td>
<td>268.086</td>
<td>1.8952</td>
<td>218</td>
<td>440</td>
<td>7.759</td>
</tr>
<tr>
<td>Low Pass Filter</td>
<td>Conceptual</td>
<td>0.66</td>
<td>47.44</td>
<td>0.5685</td>
<td>203</td>
<td>593</td>
<td>2.8</td>
</tr>
<tr>
<td></td>
<td>Behavioral</td>
<td>1.48</td>
<td>120.86</td>
<td>2.979</td>
<td>210</td>
<td>424</td>
<td>14.188</td>
</tr>
<tr>
<td></td>
<td>Structural</td>
<td>1.806</td>
<td>372.718</td>
<td>3.315</td>
<td>228</td>
<td>460</td>
<td>14.543</td>
</tr>
<tr>
<td>Band Pass Filter</td>
<td>Conceptual</td>
<td>1.82</td>
<td>267.78</td>
<td>0.5167</td>
<td>203</td>
<td>410</td>
<td>2.545</td>
</tr>
<tr>
<td></td>
<td>Behavioral</td>
<td>3.19</td>
<td>271.23</td>
<td>6.8</td>
<td>203</td>
<td>410</td>
<td>33.497</td>
</tr>
<tr>
<td></td>
<td>Structural</td>
<td>2.18</td>
<td>376.98</td>
<td>11.696</td>
<td>214</td>
<td>945</td>
<td>54.658</td>
</tr>
<tr>
<td>Band Pass Filter</td>
<td>Conceptual</td>
<td>2.32</td>
<td>227.43</td>
<td>80.83</td>
<td>2621</td>
<td>7630</td>
<td>23.2</td>
</tr>
<tr>
<td></td>
<td>Behavioral</td>
<td>3.06</td>
<td>503.94</td>
<td>68.258</td>
<td>2598</td>
<td>7632</td>
<td>26.273</td>
</tr>
<tr>
<td></td>
<td>Structural</td>
<td>3.06</td>
<td>503.94</td>
<td>68.258</td>
<td>2598</td>
<td>7632</td>
<td>26.273</td>
</tr>
<tr>
<td></td>
<td>Structural</td>
<td>3.06</td>
<td>503.94</td>
<td>68.258</td>
<td>2598</td>
<td>7632</td>
<td>26.273</td>
</tr>
<tr>
<td></td>
<td>Structural</td>
<td>3.06</td>
<td>503.94</td>
<td>68.258</td>
<td>2598</td>
<td>7632</td>
<td>26.273</td>
</tr>
<tr>
<td></td>
<td>Structural</td>
<td>3.06</td>
<td>503.94</td>
<td>68.258</td>
<td>2598</td>
<td>7632</td>
<td>26.273</td>
</tr>
<tr>
<td></td>
<td>Structural</td>
<td>3.06</td>
<td>503.94</td>
<td>68.258</td>
<td>2598</td>
<td>7632</td>
<td>26.273</td>
</tr>
<tr>
<td></td>
<td>Structural</td>
<td>3.06</td>
<td>503.94</td>
<td>68.258</td>
<td>2598</td>
<td>7632</td>
<td>26.273</td>
</tr>
<tr>
<td>RF Amplifier</td>
<td>Behavioral</td>
<td>2.44</td>
<td>448.67</td>
<td>24.289</td>
<td>1281</td>
<td>3843</td>
<td>18.961</td>
</tr>
<tr>
<td></td>
<td>Structural</td>
<td>2.17</td>
<td>331.79</td>
<td>266.95</td>
<td>123346</td>
<td>593429</td>
<td>21.62</td>
</tr>
<tr>
<td>RF Amplifier</td>
<td>Behavioral</td>
<td>1.68</td>
<td>146.87</td>
<td>2964.4</td>
<td>144023</td>
<td>886495</td>
<td>20.582</td>
</tr>
<tr>
<td></td>
<td>Structural</td>
<td>2.17</td>
<td>331.79</td>
<td>266.95</td>
<td>123346</td>
<td>593429</td>
<td>21.62</td>
</tr>
<tr>
<td>FM System</td>
<td>Structural</td>
<td>5.06</td>
<td>865.69</td>
<td>303.523</td>
<td>2691</td>
<td>10487</td>
<td>112.79</td>
</tr>
</tbody>
</table>

Table 4.1
## Execution time of SPICE models

<table>
<thead>
<tr>
<th>Model</th>
<th>Simulation Time</th>
<th>Analog Points</th>
<th>Iterations</th>
<th>Avg. Exe. Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(in sec)</td>
<td>(in msec)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Limiter</td>
<td>0.89</td>
<td>1005</td>
<td>2036</td>
<td>0.8855</td>
</tr>
<tr>
<td>Mixer</td>
<td>5.33</td>
<td>5008</td>
<td>10019</td>
<td>1.064</td>
</tr>
<tr>
<td>Low Pass Filter</td>
<td>0.86</td>
<td>1005</td>
<td>2011</td>
<td>0.8855</td>
</tr>
<tr>
<td>High Pass Filter</td>
<td>0.94</td>
<td>1005</td>
<td>2011</td>
<td>0.9353</td>
</tr>
<tr>
<td>Band Pass Filter</td>
<td>1.03</td>
<td>1005</td>
<td>2011</td>
<td>1.0248</td>
</tr>
<tr>
<td>RF Amplifier</td>
<td>1.52</td>
<td>1149</td>
<td>3087</td>
<td>1.3228</td>
</tr>
<tr>
<td>IF Amplifier</td>
<td>1</td>
<td>1014</td>
<td>2163</td>
<td>0.9862</td>
</tr>
<tr>
<td>IF Amplifier (BPF)</td>
<td>19.76</td>
<td>20005</td>
<td>40011</td>
<td>0.9877</td>
</tr>
<tr>
<td>Travis Detector</td>
<td>332.07</td>
<td>131099</td>
<td>710253</td>
<td>2.533</td>
</tr>
</tbody>
</table>

Table 4.2
<table>
<thead>
<tr>
<th>Model</th>
<th>Abstraction level</th>
<th>Avg. Exe. Time</th>
<th>Ratio</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>SEAMS (in msec)</td>
<td>SPICE (in dB)</td>
<td>(seams/spice)</td>
</tr>
<tr>
<td><strong>Limiter</strong></td>
<td>Conceptual</td>
<td>2.765</td>
<td>3.122</td>
<td>-32.04</td>
</tr>
<tr>
<td></td>
<td>Behavioral</td>
<td>7.132</td>
<td>0.8855</td>
<td>8.054</td>
</tr>
<tr>
<td></td>
<td>Structural</td>
<td>11.845</td>
<td>13.376</td>
<td>-34.09</td>
</tr>
<tr>
<td><strong>Mixer</strong></td>
<td>Conceptual</td>
<td>3.825</td>
<td>3.595</td>
<td>-10.76</td>
</tr>
<tr>
<td></td>
<td>Behavioral</td>
<td>17.828</td>
<td>1.064</td>
<td>16.755</td>
</tr>
<tr>
<td></td>
<td>Structural</td>
<td>25.234</td>
<td>23.716</td>
<td>-24.98</td>
</tr>
<tr>
<td><strong>Low Pass Filter</strong></td>
<td>Conceptual</td>
<td>8.899</td>
<td>10.04</td>
<td>-36.79</td>
</tr>
<tr>
<td></td>
<td>Behavioral</td>
<td>7.33</td>
<td>0.8855</td>
<td>8.277</td>
</tr>
<tr>
<td></td>
<td>Structural</td>
<td>7.55</td>
<td>8.526</td>
<td>-41.23</td>
</tr>
<tr>
<td><strong>High Pass Filter</strong></td>
<td>Conceptual</td>
<td>1.02</td>
<td>1.09</td>
<td>-35.98</td>
</tr>
<tr>
<td></td>
<td>Behavioral</td>
<td>7.753</td>
<td>0.9353</td>
<td>8.289</td>
</tr>
<tr>
<td></td>
<td>Structural</td>
<td>7.795</td>
<td>8.334</td>
<td>-42.05</td>
</tr>
<tr>
<td><strong>Band Pass Filter</strong></td>
<td>Conceptual</td>
<td>0.5685</td>
<td>0.5547</td>
<td>N.A</td>
</tr>
<tr>
<td></td>
<td>Behavioral</td>
<td>14.188</td>
<td>1.0248</td>
<td>13.844</td>
</tr>
<tr>
<td></td>
<td>Structural</td>
<td>14.543</td>
<td>14.191</td>
<td>-42.84</td>
</tr>
<tr>
<td><strong>RF Amplifier</strong></td>
<td>Conceptual</td>
<td>2.545</td>
<td>1.909</td>
<td>-29.84</td>
</tr>
<tr>
<td></td>
<td>Behavioral</td>
<td>33.497</td>
<td>1.3228</td>
<td>25.132</td>
</tr>
<tr>
<td></td>
<td>Structural</td>
<td>54.658</td>
<td>41.01</td>
<td>-38.97</td>
</tr>
<tr>
<td><strong>IF Amplifier</strong></td>
<td>Behavioral</td>
<td>23.2</td>
<td>23.5</td>
<td>xxx</td>
</tr>
<tr>
<td></td>
<td>Structural</td>
<td>26.273</td>
<td>0.9877</td>
<td>26.64</td>
</tr>
<tr>
<td></td>
<td>Using BPF</td>
<td>18.961</td>
<td>19.226</td>
<td>-42.64</td>
</tr>
<tr>
<td><strong>Travis Detector</strong></td>
<td>Behavioral</td>
<td>20.582</td>
<td>8.125</td>
<td>-17.41</td>
</tr>
<tr>
<td></td>
<td>Structural</td>
<td>21.62</td>
<td>2.533</td>
<td>8.535</td>
</tr>
</tbody>
</table>

Table 5.1

Summary of Results
Chapter 5

Conclusions and Future Work

5.1  Summary

The aim of this thesis was to investigate the feasibility and performance of VHDL-AMS towards modeling of systems in the high frequency region. An FM receiver system was considered for modeling and analysis. The sub-circuits of the receiver were separately modeled at different abstraction levels and were executed on SEAMS using suitable testbenches. For the same model, the results obtained from SEAMS were compared with those of SPICE using a quantitative analysis technique. The relative error in dB was computed. From the error data, it can be concluded that the models developed using VHDL-AMS are fairly accurate. The execution speed of SEAMS is found to be very low as compared to the SPICE simulator, as SEAMS is a generic simulator and SPICE is a simulator built especially for analog circuit design. Further research needs to be done in order to improve the performance of the SEAMS simulator. The results are summarized in Table 4.6.

We conclude that VHDL-AMS is indeed capable of effectively describing high frequency systems in the time-domain. Some features like frequency domain modeling and partial differential equations need to be incorporated into VHDL-AMS in order to effectively model systems such as filters, transmission lines and mixers.
5.2 Future Work

Some directions and suggestions for future work in this field are given below:

- Effective Voltage Controlled Oscillator (VCO) model needs to be developed.
- Detailed high frequency transistor models can be developed and used in the amplifier circuits.
- The receiver model can be refined further by adding impedance matching networks and tuning circuits. Other types of FM detectors can also be modeled and its performance compared.
- Noise sources need to be incorporated into all the sub-systems.
- Instead of absolute error metric, other metrics can be used for comparison.
Bibliography


Appendix A

VHDL-AMS Code

A.1 Signal Generators

A.1.1 Sine Wave

PACKAGE electricalsystem IS
  NATURE electrical IS real ACROSS real THROUGH;
  FUNCTION SIN(X:real) RETURN real;
  FUNCTION EXP(X:real) RETURN real;
END PACKAGE electricalsystem;

use work.electricalsystem.all;
ENTITY sineSource IS
  Generic(amp:real:=1.0; freq:real:=1.0);
  PORT( TERMINAL ta2,tb2 : electrical);
END sineSource;

ARCHITECTURE sinebehavior OF sineSource IS
  quantity Vsine across isine through ta2 to tb2;
BEGIN
  Vsine==(amp*sin((2.0*22.0/7.0*freq)*real(time'pos(now))*1.0e-15));
END ARCHITECTURE sinebehavior;

A.1.2 Square Wave

PACKAGE electricalSystem IS
  NATURE electrical IS real ACROSS real THROUGH;
  FUNCTION SIN(X : real) RETURN real;
  FUNCTION EXP(X : real) RETURN real;
  FUNCTION POW(X,Y : real) RETURN real;
  FUNCTION SQRT(X : real) RETURN real;
END PACKAGE electricalSystem;

use work.electricalsystem.all;
entity sqrwav is
  generic ( mag : real :=2.0;
  PORT( TERMINAL ta2,tb2 : electrical);
END sqrwav;

ARCHITECTURE squwave behavior OF sqrwav IS
  quantity Vsquare across isquare through ta2 to tb2;
BEGIN
  Vsquare==(mag*POW((1.0*time'pos(now))/freq,2))*1.0e-15);
END ARCHITECTURE squwave behavior;
A.1.3  Ramp Wave

PACKAGE electricalsystem IS
  NATURE electrical IS real ACROSS real THROUGH;
  FUNCTION SIN(X:real) RETURN real;
  FUNCTION EXP(X:real) RETURN real;
END PACKAGE electricalsystem;

use work.electricalsystem.all;
ENTITY Ramp_Source IS
  generic (slope:real:=20.0);
  PORT( TERMINAL ta2,tb2 : electrical);
END Ramp_Source;
ARCHITECTURE rampbehavior OF Ramp_Source IS
  quantity Vramp across iramp through ta2 to tb2;
BEGIN
  Vramp == slope*real(time'pos(now))*1.0e-15;
END ARCHITECTURE rampbehavior;

A.1.4  FM Wave
-- In this example: The modulating voltage is a COS wave.
-- \[ V(\text{modulating}) = V_m \cos(W_m t) \]

```vhdl
PACKAGE electricalsystem IS
    NATURE electrical IS real ACROSS real THROUGH;
    FUNCTION SIN(X:real) RETURN real;
    FUNCTION COS(X:real) RETURN real;
END PACKAGE electricalsystem;

use work.electricalsystem.all;

ENTITY fm_source IS
    generic(c_freq:real:=100.0e6; -- carrier frequency
             s_freq:real:=25.0e3; -- modulating(signal) frequency
             V_fm:real:=1.0 -- Peak Voltage of FM Signal
             );
    PORT(TERMINAL fm_out,fm_gnd : electrical);
END fm_source;

ARCHITECTURE fm_behavior OF fm_source IS
    quantity V_fm_signal across i_fm_signal through fm_out to fm_gnd;
BEGIN
    -- the max. freq. deviation is 75.0kHz
    V_fm_signal == (V_fm*sin((2.0*22.0/7.0*c_freq*real(time'pos(now))*1.0e-15)+(75.0e3/s_freq*sin(2.0*22.0/7.0*s_freq*real(time'pos(now))*1.0e-15))));
END ARCHITECTURE fm_behavior;

A.1.5 AM Wave

```vhdl
PACKAGE electricalsystem IS
    NATURE electrical IS real ACROSS real THROUGH;
    FUNCTION SIN(X:real) RETURN real;
    FUNCTION COS(X:real) RETURN real;
END PACKAGE electricalsystem;

use work.electricalsystem.all;

ENTITY am_Source IS
    generic(c_freq:real:=1.0; s_freq:real:=1.0);
    PORT( TERMINAL ta2,tb2 : electrical);
END am_Source;

ARCHITECTURE am_behavior OF am_Source IS
    quantity V_am across i_am through ta2 to tb2;
```
A.2. Limiter

A.2.1 Conceptual Level Model

PACKAGE electricalsystem IS
         nature electrical IS real across real through;
         FUNCTION SIN (X : real ) RETURN real;
         FUNCTION exp (X : real ) RETURN real;
END PACKAGE electricalsystem;

use work.electricalsystem.all;

entity limiter_conceptual is
         generic(V_limit:real:=1.5);
         port(terminal Signal_in,Signal_out: electrical);
end entity limiter_conceptual;

architecture lim_conceptual of limiter_conceptual is

         quantity v_in across i_in through signal_in to electrical’reference;
         quantity v_op through signal_out to electrical’reference;

begin

         v_in == i_in*1.0e6;
         if (v_in>=0.0 and v_in >= v_limit) use
             v_op == v_limit*1.0;
         elsif (v_in>=0.0 and v_in < v_limit) use
             v_op == v_in*1.0;
         elsif (v_in<0.0 and v_in < -v_limit) use
             v_op == -1.0*v_limit;
         elsif (v_in<0.0 and v_in >= -v_limit) use
             v_op == v_in*1.0;
         end use;

end architecture lim_conceptual;

A.2.2 Behavioral Level Model

PACKAGE electricalsystem IS
         nature electrical IS real ACROSS real THROUGH;
         FUNCTION SIN(X:real) RETURN real;
         FUNCTION exp(X:real) RETURN real;
END PACKAGE electricalsystem;

use work.electricalsystem.all;
entity limiter is
  generic (lim:real:=1.0);
  port (terminal v_in,v_out :electrical);
end entity limiter;

architecture behav of limiter is

terminal t1,t2,t3 :electrical;

--> Constant for the diode's
--------------------------
constant k:real := 0.02586; -- thermal voltage
constant iss:real := 1.8104e-15;
constant gmin:real := 1.0e-12;

--> Quantities for the diode's
-----------------------------
quantity vd1 across id1 through t1 to t2;
quantity vd2 across id2 through t3 to v_out;

--> Quantities for the voltage source
-------------------------------------
quantity V_volt1 across i_volt1 through t2 to electrical'reference;
quantity V_volt2 across i_volt2 through electrical'reference to t3;

--> Quantities for the resistances
-------------------------------
quantity v_r1 across i_r1 through V_in to T1;
quantity v_r2 across i_r2 through T1 to V_out;

BEGIN

--> D1
  if (vd1 >= (-5.0*k)) use
    id1 == iss * (exp(vd1/k)-1.0) + vd1*gmin;
  elsif (vd1<-5.0*k) use
    id1 == -1.0*iss + vd1*gmin;
  end use;

--> D2
  if (vd2 >= (-5.0*k)) use
    id2 == iss * (exp(vd2/k)-1.0) + vd2*gmin;
  elsif (vd2<-5.0*k) use
    id2 == -1.0*iss + vd2*gmin;
  end use;

--> V1
  V_volt1 == (lim);

--> V2
  V_volt2 == (lim);

--> Res r1 and r2
  V_r1 == i_r1*10.0;
  V_r2 == i_r2*10.0;
end architecture behav;

A.2.3 Structural Level Model
PACKAGE electricalsystem IS
    NATURE electrical IS real ACROSS real THROUGH;
    FUNCTION SIN(X:real) RETURN real;
    FUNCTION EXP(X:real) RETURN real;
END PACKAGE electricalsystem;

-- Include diode(A.18) code here
-------------- Voltage Source --------------------------
use work.electricalsystem.all;
ENTITY voltSource IS
    generic( amp:real:=1.0);
    PORT( TERMINAL ta2,tb2 : electrical);
END voltSource;
ARCHITECTURE voltbehavior OF voltSource IS
    quantity V_volt across i_volt through ta2 to tb2;
BEGIN
    V_volt ==amp;
END ARCHITECTURE voltbehavior;

-------------- LIMITER ----------------------------
use work.electricalsystem.all;
entity limiter is
    generic (lim:real:=1.0);
    port (terminal v_in,v_out : electrical);
end entity limiter;
architecture struct of limiter is
    --> Components
diode 
    generic (
        Isat : real := 1.0e-14; -- saturation current
        n : real := 1.0; -- emission coefficient
        bv : real := 15.0; -- reverse breakdown voltage
        ibv : real := 1.0e-3; -- Breakdown current
        rds : real := 0.0 -- Ohmic resistance
    );
    port (terminal pos, neg : electrical);
end component;
for all : diode use entity work.Diode(behav);
voltsource
    generic( amp:real:=1.0);
    PORT( TERMINAL ta2,tb2 : electrical);
end component;
for all: voltsource use entity work.voltsource(voltbehavior);
terminal t1,t2,t3 : electrical;
quantity v_r1 across i_r1 through V_in to T1;
quantity v_r2 across i_r2 through T1 to V_out;
BEGIN
    D1 : diode port map(T2,T1);
D2 : diode port map(V_out,T3);
V1 : voltsource generic map(lim)
    port map(electrical’reference,T2);
V2 : voltsource generic map(lim)
    port map(t3,electrical’reference);
    V_r1 == i_r1*10.0;
    V_r2 == i_r2*10.0;
end architecture struct;

A.3 Mixer

A.3.1 Conceptual Model

PACKAGE electricalsystem IS
    NATURE electrical IS real ACROSS real THROUGH;
    FUNCTION SIN(X:real) RETURN real;
    FUNCTION EXP(X:real) RETURN real;
END PACKAGE electricalsystem;

use work.electricalsystem.all;

entity mixer_conceptual is
    port(terminal rf_in,rf_gnd,lo_osc,lo_gnd,mixer_op,op_gnd
        :electrical);
end entity mixer_conceptual;

architecture mixer_conceptual_behav of mixer_conceptual is
    CONSTANT R_rf:real:=500.0;
    CONSTANT R_lo:real:=500.0;  -->
    CONSTANT R_op:real:=500.0;  -->
    CONSTANT gain:real:=4.0e-4;

    quantity v_rf across i_rf through rf_in to rf_gnd;
    quantity v_lo across i_lo through lo_osc to lo_gnd;
    quantity v_out across i_out through mixer_op to op_gnd;

begin
    v_rf ==i_rf*R_rf;
    v_lo ==i_lo*R_lo;
    v_out==v_rf*v_lo*gain;
end architecture mixer_conceptual_behav;

A.3.2 Behavioral Model

PACKAGE electricalSystem IS
    NATURE electrical IS real ACROSS real THROUGH;
    FUNCTION SIN (X : real ) RETURN real;
    FUNCTION Sqrt (X : real ) RETURN real;
    FUNCTION COS (X : real ) RETURN real;
    FUNCTION exp (X : real ) RETURN real;
END PACKAGE electricalSystem;
use work.electricalsystem.all;

entity mixer_behav is
  port(terminal rf_in,rf_gnd,lo_osc,lo_gnd,mixer_op,mixer_gnd :electrical);
end entity mixer_behav;

architecture behav of mixer_behav is

constant k1 : real := 0.9999; -- coupling factor
constant k2 : real := 0.9999; -- coupling factor
constant k3 : real := 0.9999; -- coupling factor
constant lp : real := 10.0e-3; -- primary inductance
constant ls : real := 80.0e-3; -- secondary inductance
constant tap : real := 0.5; -- tapping point

terminal t1,t2,t3,t4:electrical;
terminal t11,t12,t13,t14:electrical;
terminal pos1,pos2,pos3,pos4 :electrical;

quantity m1,m2,m3:real;

quantity vin_1 across iin_1 through t1 to rf_gnd;
quantity vout_1 across iout_1 through t2 to t3;
quantity vout_2 across iout_2 through t3 to t4;

quantity vr1 across ir1 through rf_in to t1;
quantity vr2 across ir2 through t2 to pos1;
quantity vr3 across ir3 through t3 to mixer_gnd;
quantity vr4 across ir4 through t4 to pos3;

quantity vin_11 across iin_11 through t11 to lo_gnd;
quantity vout_11 across iout_11 through t12 to t13;
quantity vout_12 across iout_12 through t13 to t14;

quantity vr11 across ir11 through lo_osc to t11;
quantity vr12 across ir12 through t12 to pos4;
quantity vr13 across ir13 through t13 to mixer_op;
quantity vr14 across ir14 through t14 to pos2;

-- Constants for the diode
constant k:real:=0.02586; -- thermal voltage
constant iss:real:=1.8104e-15;
constant gmin:real:=1.0e-12;

quantity vd1 across id1 through pos1 to pos2;
quantity vd2 across id2 through pos2 to pos3;
quantity vd3 across id3 through pos3 to pos4;
quantity vd4 across id4 through pos4 to pos1;

begin
  brk : break vd1 => 1.0,vd2=>0.0,vd3=>0.0,vd4=>0.0;
  brk1 : break iin_11=>0.0, iout_12=>0.0, iout_11=>0.0;
  brk2 : break iin_1=>0.0, iout_2=>0.0, iout_1=>0.0;

  m1=k1*sqrt(lp*ls*(1.0-tap));
  m2=k2*sqrt(lp*ls*tap);
  m3=k3*sqrt(ls*tap*ls*(1.0-tap));
vr1 == ir1 *10.0;
vr2 == ir2 *10.0;
vr3 == ir3 *10.0;
vr4 == ir4 *10.0;
vr11 == ir11*10.0;
vr12 == ir12*10.0;
vr13 == ir13*10.0;
vr14 == ir14*10.0;

ipl1 : vin_1 == lp*iin_1'dot + m2*iout_2'dot+ m1*iout_1'dot;
up1 : vout_1 == ls*(1.0-tap)*iout_1'dot+ m1*iin_1'dot + m3*iout_2'dot;
down1 : vout_2 == ls*tap*iout_2'dot + m2*iin_1'dot + m3*iout_1'dot;

ip : vin_11 == lp*iin_11'dot + m2*iout_12'dot+ m1*iout_11'dot;
up : vout_11 == ls*(1.0-tap)*iout_11'dot+ m1*iin_11'dot + m3*iout_12'dot;
down : vout_12 == ls*tap*iout_12'dot + m2*iin_11'dot + m3*iout_11'dot;

if (vd1 >= (-5.0*k)) use
  id1 == iss * (exp(vd1/k)-1.0) + vd1*gmin;
elsif (vd1<-5.0*k) use
  id1 == -1.0*iss + vd1*gmin;
end use;

if (vd2 >= (-5.0*k)) use
  id2 == iss * (exp(vd2/k)-1.0) + vd2*gmin;
elsif (vd2<-5.0*k) use
  id2 == -1.0*iss + vd2*gmin;
end use;

if (vd3 >= (-5.0*k)) use
  id3 == iss * (exp(vd3/k)-1.0) + vd3*gmin;
elsif (vd3<-5.0*k) use
  id3 == -1.0*iss + vd3*gmin;
end use;

if (vd4 >= (-5.0*k)) use
  id4 == iss * (exp(vd4/k)-1.0) + vd4*gmin;
elsif (vd4<-5.0*k) use
  id4 == -1.0*iss + vd4*gmin;
end use;
end architecture behav;

A.3.3 Structural Model

PACKAGE electricalSystem IS
  NATURE electrical IS real ACROSS real THROUGH;
  FUNCTION SIN (X : real ) RETURN real;
  FUNCTION Sqrt (X : real ) RETURN real;
  FUNCTION COS (X : real ) RETURN real;
  FUNCTION exp (X : real ) RETURN real;
END PACKAGE electricalSystem;

-- Insert tapped transformer(A.11.2) and diode(A.18) code here
use work.electricalsystem.all;

entity mixer_struct is
  port(terminal rf_in,rf_gnd,lo_osc,lo_gnd,mixer_op,mixer_gnd :electrical);
end entity mixer_struct;

architecture struct of mixer_struct is

  COMPONENT TRANSFORMER IS
    generic (
      k1 : real := 0.9999; -- coupling factor
      k2 : real := 0.9999; -- coupling factor
      k3 : real := 0.9999; -- coupling factor
      lp : real := 10.0e-3; -- primary inductance
      ls : real := 10.0e-3; -- secondary inductance
      tap : real := 0.5); -- tapping point
    port (terminal v_in,gnd_in,t_up,t_mid,t_down : electrical);
  end COMPONENT;

  for all: transformer use entity work.transformer(trans_behav);

  component diode
    generic (
      Isat : real := 1.0e-14; -- saturation current
      n : real := 1.0; -- emission coefficient
      bv : real := 15.0; -- reverse breakdown voltage
      ibv : real := 1.0e-3; -- Breakdown current
      rds : real := 0.0 -- Ohmic resistance
    );
    port (terminal pos, neg : electrical);
  end component;

  for all : diode use entity work.Diode(behav);

  terminal t1,t2,t3,t4 :electrical;

begin

  trans1: transformer generic map(0.9999,0.9999,0.9999,10.0e-3,80.0e-3,0.5)
    port map(rf_in,rf_gnd,t1,mixer_gnd,t3);
  trans2: transformer generic map(0.9999,0.9999,0.9999,10.0e-3,80.0e-3,0.5)
    port map(lo_osc,lo_gnd,t4,mixer_op,t2);

  d1 : diode port map(t1,t2);
  d2 : diode port map(t2,t3);
  d3 : diode port map(t3,t4);
  d4 : diode port map(t4,t1);

end architecture struct;

A.3.4  Active Mixer Using a FET

PACKAGE electricalsystem IS
  NATURE electrical IS real ACROSS real THROUGH;
  FUNCTION SIN(X:real) RETURN real;
  FUNCTION EXP(X:real) RETURN real;
  FUNCTION SQRT(X:real) RETURN real;
END PACKAGE electricalsystem;
use work.electricalsystem.all;

entity mixer_fet_struct is
  generic (v_cgs : real := -0.0298; v_cc : real := -0.0298; v_cgd : real := 9.0; v_cds : real := 8.982; v_c1 : real := 0.0; v_c2 : real := 0.0; v_c3 : real := 0.03; tap_r1 : real := 0.5; tap_r2 : real := 0.5);
  port (terminal rf_in, rf_gnd, lo_in, lo_gnd, mixer_op, mixer_gnd, V_source : electrical);
end entity mixer_fet_struct;

architecture mixer_struct of mixer_fet_struct is
  --> Components are defined here
  component fet_struct is
    generic (v_cgs : real := -0.0298; v_cc : real := -0.0298; v_cgd : real := 9.0; v_cds : real := 8.982);
    port (terminal source, drain, gate : electrical);
  end component;
  for all : fet_struct use entity work.fet_Struct(struct);
  component capacitor is
    generic (cap : real := 1.0; v_init : real := 0.0);
    port (terminal c_in, c_out : electrical);
  end component;
  for all : capacitor use entity work.capacitor(behav);
  component resistor is
    generic (res : real := 1.0);
    port (terminal r_in, r_out : electrical);
  end component;
  for all : resistor use entity work.resistor(behav);
  component TRANSFORMER IS
    generic (k : real := 0.9999; -- coupling factor
             lp : real := 10.0e-3; -- primary inductance
             ls : real := 10.0e-3); -- secondary inductance
    port (terminal t1, t2, t3, t4 : electrical);
  end component;
  for all : transformer use entity work.transformer(behavior);
  terminal t1, t2, t3, t4, t5, t6, t7, t8 : electrical;
begin

  C1 : capacitor generic map (0.1e-6, v_c1)
      port map (rf_in, T1);
  C2 : capacitor generic map (0.1e-6, v_c2)
      port map (lo_in, T4);
  C3 : capacitor generic map (0.1e-6, v_c3)
      port map (T6, T8);
  Rl_a : resistor generic map ((1.0 - tap_r1) * 10.0e3)
        port map (T1, T2);
  Rl_b : resistor generic map (tap_r1 * 10.0e3)
        port map (T2, electrical‘reference);
  R2 : resistor generic map (100.0e3)
port map(T2,T3);
R3_a :resistor generic map((1.0-tap_r2)*10.0e3)
    port map(T4,T5);
R3_b :resistor generic map(tap_r2*10.0e3)
    port map(T5,electrical'reference);
R4 :resistor generic map(100.0e3)
    port map(T4,T3);
R5 :resistor generic map(6.8e3)
    port map(T6,electrical'reference);
JFET :fet_struct generic map(-0.0298,-0.0298,-9.0,8.982)
    port map(T6,T7,T3);
trans:transformer generic map(0.9999,10.0e-3,30.0e-3)
    port map(T8,electrical'reference,mixer_op,mixer_gnd);
R_sh :resistor generic map(1.0e-30)
    port map(T7,V_source);
end architecture mixer_struct;

A.4 RF Amplifier

A.4.1 Conceptual Level

PACKAGE electricalsystem IS
    NATURE electrical IS real ACROSS real THROUGH;
    FUNCTION SIN(X:real) RETURN real;
    FUNCTION EXP(X:real) RETURN real;
END PACKAGE electricalsystem;

use work.electricalsystem.all;
ENTITY rf_amp_conceptual is
    GENERIC(amp:real:=12400.0);
    PORT(TERMINAL signal_in,gnd_in,signal_out,gnd_out: electrical);
END ENTITY rf_amp_conceptual;

ARCHITECTURE rf_amp_conceptual_behav of rf_amp_conceptual is
    CONSTANT r_in :real:=1.0e6;
    quantity v_in across i_in through signal_in to gnd_in;
    quantity v_gn across i_gn through signal_in to signal_out;
BEGIN
    v_in == i_in*R_in;
    V_gn == amp*v_in;
END ARCHITECTURE rf_amp_conceptual_behav;

A.4.2 Structural Model of a 2-stage amplifier

PACKAGE electricalsystem IS
    NATURE electrical IS real ACROSS real THROUGH;
    FUNCTION SIN(X:real) RETURN real;
    FUNCTION EXP(X:real) RETURN real;
END PACKAGE electricalsystem;

-- Include res(A.19), cap(A.20) and transistor(A.16.1) code here
use work.electricalsystem.all;

entity audio_pre_amp_struct is
  generic(tap_op:real:=0.5);
  port (terminal audio_in,aud_in_gnd,audio_out,
    aud_out_gnd,V_source:electrical);
end entity audio_pre_amp_struct;

architecture audio_struct of audio_pre_amp_struct is
  --> Components are defined here
  component capacitor is
    generic(cap :real:=1.0);
    port(terminal c_in,c_out: electrical);
  end component;
  for all: capacitor use entity work.capacitor(behav);

  component resistor is
    generic(res :real:=1.0 );
    port(terminal r_in,r_out: electrical);
  end component;
  for all: resistor use entity work.resistor(behav);

  component trans_t_eq_struct is
    port( terminal Emitter,Base,Collector : electrical);
  end component;
  for all: trans_t_eq_struct use entity work.trans_t_eq_struct(trans_behav);

  terminal t1,t2,t3,t4,t5,t6,t7,t8 :electrical;
BEGIN
  R1: resistor generic map(47.0e3)
      port map(v_source,T1);
  R2: resistor generic map(10.0e3)
      port map(t1,electrical'reference);
  R3: resistor generic map(4.7e3)
      port map(v_source,T3);
  R4: resistor generic map(1.0e3)
      port map(t2,electrical'reference);
  R5: resistor generic map(47.0e3)
      port map(v_source,T5);
  R6: resistor generic map(10.0e3)
      port map(t4,electrical'reference);
  R7: resistor generic map(4.7e3)
      port map(v_source,T6);
  R8: resistor generic map(1.0e3)
      port map(v_source,T5);
  Ra: resistor generic map(3.0e3)
      port map(t5,electrical'reference);
  Rb: resistor generic map(300.0)
      port map(t8,audio_out);
  C1: capacitor generic map(1.0e-6)
      port map(audio_out,electrical'reference);
  C2: capacitor generic map(100.0e-6)
      port map(t2,electrical'reference);
  C3: capacitor generic map(1.0e-6)
      port map(t3,t4);
  C4: capacitor generic map(100.0e-6)
      port map(t5,electrical'reference);
C5: capacitor generic map(1.0e-6)
    port map(t6,t8);

bjt1 : trans_t_eq_struct port map(t2,t1,T3);
bjt2 : trans_t_eq_struct port map(t5,t4,T6);

end architecture audio_struct;

A.5 IF Amplifier

PACKAGE electricalsystem IS
    NATURE electrical IS real ACROSS real THROUGH;
    FUNCTION SIN(X:real) RETURN real;
    FUNCTION EXP(X:real) RETURN real;
    FUNCTION SQRT(X:real) RETURN real;
END PACKAGE electricalsystem;

-- Include the transistor(A.16.1),res(A.19), cap(A.20) and tuned
-- transformer(A.11.3)code here.
use work.electricalsystem.all;

entity if_amp is
    port(terminal v_in,v_in_gnd,v_out,v_out_gnd:electrical);
end entity if_amp;

architecture if_amp_behav of if_amp is

--> COMPONENTS ARE DECLARED HERE

COMPONENT trans_t_eq_struct is
    port( terminal emitter,base,collector : electrical);
end component;
for all : trans_t_eq_struct use entity work.trans_t_eq_struct
    (trans_behav);

component resistor is
    generic(res :real:=1.0 );
    port(terminal r_in,r_out: electrical);
end component;
for all: resistor use entity work.resistor(behav);

component capacitor is
    generic(cap :real:=1.0;v_init:real:=0.0);
    port(terminal c_in,c_out: electrical);
end component;
for all: capacitor use entity work.capacitor(behav);

component tuned_transformer IS
    generic (freq_fm : real := 1.0);
    port (terminal Signal_in,in_gnd,Signal_out,out_gnd : electrical);
end component;
for all : tuned_transformer use entity work.tuned_transformer
    (behav);

terminal t1,t2,t3,V_source :electrical;
terminal temp1,temp2:electrical;
quantity \( V_{\text{volt}} \) across \( i_{\text{volt}} \) through \( V_{\text{source}} \) to electrical reference;

begin

trans_t_eq_struct port map(emitter=>T2, Base=>T1, collector=>T3);
res_b1: resistor generic map(47.0e3)
port map(V_source, T1);
res_b2: resistor generic map(10.0e3)
port map(T1, v_in_gnd);
res_re: resistor generic map(10.0e3)
port map(T2, v_in_gnd);
cap_ce: capacitor generic map(cap=>100.0e-6, v_init=>0.0)
port map(T2, v_in_gnd);
cap_c1: capacitor generic map(cap=>1.0e-6, v_init=>0.0)
port map(v_in, T1);
FM_AM: tuned_transformer generic map(freq_fm=>10.7e6)--IF freq.
port map(V_source, T3, v_out, v_out_gnd);

v_src: v_volt==5.0;

end architecture if_amp_behav;

A.6 FM Detector

A.6.1 Travis Detector

PACKAGE electricalSystem IS
NATURE electrical IS real ACROSS real THROUGH;
FUNCTION SIN (X : real ) RETURN real;
FUNCTION COS (X : real ) RETURN real;
FUNCTION EXP (X : real ) RETURN real;
FUNCTION SQRT (X : real) RETURN real;
END PACKAGE electricalSystem;

--Insert code for tuned tapped transformer(A.11.4) and Diode(A.18)

use work.electricalsystem.all;

entity travis_demodulator is
port(terminal Signal_in, Audio_out: electrical);
end entity travis_demodulator;

architecture struct of travis_demodulator is

--> components

component tuned_tapped IS
generic (freq_upsec: real := 1.0; freq_downsec: real:=1.0);
port (terminal Signal_in, in_gnd, T_up, T_mid, T_down: electrical);
end component;
for all : tuned_tapped use entity work.tuned_tapped(behav);

component diode

generic (Isat : real := 1.0e-14; -- saturation current
n : real := 1.0; -- emmission coefficient
bv : real := 15.0; -- reverse breakdown voltage
ibv : real := 1.0e-3; -- Breakdown current

end component diode;


rds : real := 0.0 -- Ohmic resistance
);  
end component;
for all : diode use entity work.Diode(behav);

terminal T1,t2,T3: electrical;

quantity v_out across i_out through audio_out to electrical'reference;
quantity v_r1 across i_r1 through audio_out to t2;
quantity i_c1 through audio_out to t2;
quantity v_r2 across i_r2 through electrical'reference to t2;
quantity i_c2 through electrical'reference to t2;

begin

FM_AM : tuned_tapped generic map(10.7816e6,10.6184e6)
port map(signal_in,electrical'reference,
t1,t2,t3);

D1 : diode port map(T1,audio_out);
D2 : diode port map(T3,electrical'reference);

v_out == v_r1 +v_r2;

v_r1 == i_r1*500.0e3;

v_r2 == i_r2*500.0e3;
i_c1 == 100.0e-12*v_r1'dot;
i_c2 == 100.0e-12*v_r2'dot;

end struct;

A.6.2 Slope Detector

PACKAGE electricalSystem IS

NATURE electrical IS real ACROSS real THROUGH;
FUNCTION SIN (X : real ) RETURN real;
FUNCTION COS (X : real ) RETURN real;
FUNCTION EXP (X : real ) RETURN real;
FUNCTION SQRT (X : real) RETURN real;
END PACKAGE electricalSystem;

-- include diode(A.18) and FM_2_AM_Converter(A.10) code here

use work.electricalSystem.all;

entity slope_demodulator is
port (terminal FM_in,audio_out:electrical);
end entity slope_demodulator;

architecture struct of slope_demodulator is

-- Components

cOMPONENT FM_2_AM_Converter IS
generic (freq_fm : real := 1.0);
port (terminal Signal_in, Signal_out : electrical);
end component;
for all : FM_2_AM_Converter use entity work.FM_2_AM_Converter(behav);

component diode
generic (  
  Isat : real := 1.0e-14; -- saturation current  
  n : real := 1.0; -- emission coefficient  
  bv : real := 15.0; -- reverse breakdown voltage  
  ibv : real := 1.0e-3; -- Breakdown current  
  rds : real := 0.0  -- Ohmic resistance
)
port (terminal pos, neg : electrical);
end component;
for all : diode use entity work.Diode(behav);

CONSTANT RES: real:=100.0e3;
CONSTANT CAP: real:=100.0e-12; -- RC = 10e-5

Terminal t1,t2,t3: electrical;
quantity v_res1 across i_res1 through audio_out to electrical'reference;
quantity i_cap1 through audio_out to electrical'reference;

BEGIN

  FMAM : FM_2_AM_CONVERTER generic map(10.7816e6)
    port map(FM_in,T1);
  Diodel : diode port map(audio_out,T1);
  v_res1 == i_res1 * RES;
  i_cap1 == CAP * v_res1'dot;
end struct;

A.7 Filter

A.7.1 Structural Level model of a Active LPF using ideal op-amp

PACKAGE electricalsystem IS
  NATURE electrical IS real ACROSS real THROUGH;
  FUNCTION SIN(X:real) RETURN real;
  FUNCTION EXP(X:real) RETURN real;
END PACKAGE electricalsystem;

--include res(A.19),cap(A.20) and ideal op-amp(A.12.1) code here
use work.electricalsystem.all;

entity low_pass_filter is
  generic(freq:real:=100.0e6);
    port(terminal V_in,V_out :electrical);
end entity low_pass_filter;

architecture struct of low_pass_filter is

  --> Components
    component capacitor is

generic(cap : real := 1.0; v_init : real := 0.0);
port (terminal c_in, c_out: electrical);
end component;
for all: capacitor use entity work.capacitor(behav);

component resistor is
generic(res : real := 1.0);
port (terminal r_in, r_out: electrical);
end component;
for all: resistor use entity work.resistor(behav);

component op_amp is
port (terminal inverting_ip, non_inverting_ip, output : electrical);
end component;
for all: op_amp use entity work.op_amp(struct);

terminal T1, T2, T3: electrical;
BEGIN
Ra : resistor generic map(res=>1000.0)
    port map(V_in, T1);
Rb : resistor generic map(res=>1000.0)
    port map(T1, T2);
Ca : capacitor generic map(cap=>(1.59155e-4/freq), v_init=>0.0)
    port map(T1, V_out);
Cb : capacitor generic map(cap=>(1.59155e-4/freq), v_init=>0.0)
    port map(T2, electrical'reference);
LM709: op_amp port map(non_inverting_ip=>T2, inverting_ip=>T3,
                          output=>V_out);
R1 : resistor generic map(res=>586.0)
    port map(v_out, T3);
R2 : resistor generic map(res=>1000.0)
    port map(T3, electrical'reference);

end architecture struct;

A.7.2 Structural Level model of a Active HPF using ideal op-amp

PACKAGE electricalsystem IS
    NATURE electrical IS real ACROSS real THROUGH;
    FUNCTION SIN(X: real) RETURN real;
    FUNCTION EXP(X: real) RETURN real;
END PACKAGE electricalsystem;

-- insert res(A.19), cap(A.20), ideal op-amp (A.12.1) code here
use work.electricalsystem.all;

entity high_pass_filter is
generic(freq: real := 100.0e6);
    port (terminal V_in, V_out: electrical);
end entity high_pass_filter;

architecture struct of high_pass_filter is
--> Components
    component capacitor is
generic(cap : real := 1.0; v_init : real := 0.0);
    component resistor is
generic(res : real := 1.0);
    component op_amp is
port (terminal inverting_ip, non_inverting_ip, output : electrical);
end component;
port (terminal \texttt{c\_in}, \texttt{c\_out}: electrical);
end component;
for all: capacitor use entity work.capacitor(behav);

component resistor is
generic(res :real:=1.0);
port (terminal \texttt{r\_in}, \texttt{r\_out}: electrical);
end component;
for all: resistor use entity work.resistor(behav);

component op\_amp is
port (terminal \texttt{inverting\_ip}, \texttt{non\_inverting\_ip}, \texttt{output}: electrical);
end component;
for all: op\_amp use entity work.op\_amp(struct);

terminal \texttt{T1}, \texttt{T2}, \texttt{T3}: electrical;

BEGIN

Ra : resistor generic map(res=>1000.0)
    port map(T1,V\_out);
Rb : resistor generic map(res=>1000.0)
    port map(T2,electrical'reference);
Ca : capacitor generic map(cap=>(1.591549431e-4/freq),
    v_init=>0.0)
    port map(V\_in,T1);
Cb : capacitor generic map(cap=>(1.591549431e-4/freq),
    v_init=>0.0)
    port map(T1,T2);
LM709: op\_amp    port map(non\_inverting\_ip=>T2,inverting\_ip=>T3,
    output=>V\_out);

R1 : resistor generic map(res=>586.0)
    port map(v\_out,T3);
R2 : resistor generic map(res=>1000.0)
    port map(T3,electrical'reference);

end architecture struct;

A.7.3 Structural Level model of a Active BPF using ideal op-amp

PACKAGE electricalsystem IS
    NATURE electrical IS real ACROSS real THROUGH;
    FUNCTION SIN(X:real) RETURN real;
    FUNCTION EXP(X:real) RETURN real;
END PACKAGE electricalsystem;

-- Include Active LPF(A.7.1) and Active HPF(A.7.2) code
use work.electricalsystem.all;

entity band_pass_filter is
generic(f\_min:real:=98.5e6; f\_max:real:=108.5e6);
    port (terminal \texttt{V\_in}, \texttt{V\_out}: electrical);
end entity band_pass_filter;

architecture struct of band_pass_filter is

--> Components are defined here

    component low_pass_filter is
A.7.4 Structural Level model of a Active LPF using op-amp 741

PACKAGE electricalsystem IS
    NATURE electrical IS real ACROSS real THROUGH;
    FUNCTION SIN(X:real) RETURN real;
    FUNCTION EXP(X:real) RETURN real;
END PACKAGE electricalsystem;

-- Include code for res(A.19), cap(A.20), trans(A.16.3 &A.16.2)
-- and op-map(A.12.3)

use work.electricalsystem.all;
ENTITY voltSource IS
    generic(amp:real:=22.0);
    PORT( TERMINAL ta2,tb2 : electrical);
END voltSource;

ARCHITECTURE voltbehavior OF voltSource IS
    terminal temp: electrical;
    quantity V_volt across i_volt through temp to tb2;
    quantity V_drop across i_drop through temp to ta2;
BEGIN
    V_volt == amp;
    V_drop == i_drop*100.0;
END ARCHITECTURE voltbehavior;

-- Structural Model Of a 1'st Order Low pass Butterworth Filter -----
---------------------------------------------------------------------

use work.electricalsystem.all;
entity low_pass_filter is
    generic(freq:real:=100.0e6);
    port (terminal V_in,V_out :electrical);
end entity low_pass_filter;
architecture struct of low_pass_filter is

--> Components
component capacitor is
generic(cap :real:=1.0);
port(terminal c_in,c_out: electrical);
end component;
for all: capacitor use entity work.capacitor(behav);

component resistor is
generic(res :real:=1.0 );
port(terminal r_in,r_out: electrical);
end component;
for all: resistor use entity work.resistor(behav);

component op_amp is
port(terminal pos_ip,neg_ip,op,V_pos,V_neg: electrical);
end component;
for all: op_amp use entity work.op_amp(struct);

component voltSource IS
generic(amp:real:=22.0);
PORT( TERMINAL ta2,th2 : electrical);
END component;
for all: voltsource use entity work.voltsource(voltbehavior);

terminal T1,T2,T3,v_p,v_n:electrical;

BEGIN
Ra : resistor generic map(res=>1000.0)
    port map(V_in,T1);
Ca : capacitor generic map(cap=>(1.59155e-4/freq))
    port map(T1,electrical'reference);
LM741: op_amp port map(pos_ip=>T1,neg_ip=>T3,op=>V_out,
    V_pos=>v_p,V_neg=>v_n);
R1 : resistor generic map(res=>586.0)
    port map(v_out,T3);
R2 : resistor generic map(res=>1000.0)
    port map(T3,electrical'reference);
vp : voltSource generic map(5.0)
    port map(v_p,electrical'reference);
vn : voltSource generic map(-5.0)
    port map(v_n,electrical'reference);

end architecture struct;

A.7.5 Structural Level model of a Active HPF using op-amp 741

PACKAGE electricalsystem IS
    NATURE electrical IS real ACROSS real THROUGH;
    FUNCTION SIN(X:real) RETURN real;
    FUNCTION EXP(X:real) RETURN real;
END PACKAGE electricalsystem;
include code for res(A.19), cap(A.20), trans(A.16.3 & A.16.2)
and op-map(A.12.3)

-- Structural Model Of a 1'st Order High pass Butterworth Filter ------
------------------------------------------------------------------------
use work.electricalsystem.all;

entity high_pass_filter is
generic(freq:real:=100.0e6);
port(terminal V_in,V_out :electrical);
end entity high_pass_filter;

architecture struct of high_pass_filter is
    --> Components
    component capacitor is
generic(cap :real:=1.0);
port(terminal c_in,c_out: electrical);
end component;
    for all: capacitor use entity work.capacitor(behav);

    component resistor is
generic(res :real:=1.0 );
port(terminal r_in,r_out: electrical);
end component;
    for all: resistor use entity work.resistor(behav);

    component op_amp is
port(terminal pos_ip,neg_ip,op,V_pos,V_neg: electrical);
end component;
    for all: op_amp use entity work.op_amp(struct);

    component voltSource IS
generic(amp:real:=22.0);
PORT( TERMINAL ta2,tb2 : electrical);
END component;
    for all: voltsource use entity work.voltsource(voltbehavior);

    terminal T1,T2,T3,v_p,v_n:electrical;
BEGIN
Ra : resistor generic map(res=>1000.0)
    port map(T1,electrical'reference);
Ca : capacitor generic map(cap=>(1.59155e-4/freq))
    port map(V_in,T1);
LM741: op_amp port map(pos_ip=>T1,neg_ip=>T3,op=>V_out,
                        V_pos=>v_p,V_neg=>v_n);
R1 : resistor generic map(res=>586.0)
    port map(v_out,T3);
R2 : resistor generic map(res=>1000.0)
    port map(T3,electrical'reference);
vp : voltsource generic map(5.0)
    port map(v_p,electrical'reference);
vn : voltsource generic map(-5.0)
    port map(v_n,electrical'reference);
end architecture struct;
### A.8 Voltage Controlled Oscillator

```plaintext
PACKAGE electricalsystem IS
  NATURE electrical IS real ACROSS real THROUGH;
  FUNCTION SIN(X:real) RETURN real;
  FUNCTION EXP(X:real) RETURN real;
END PACKAGE electricalsystem;

use work.electricalsystem.all;
entity VCO is
generic(
  amp: real := 1.0; -- Amplitude of the VCO
  fc : real := 1.0E6; -- VCO frequency at Vc
  df : real := 75.0e3; -- [Hz/V], frequ. Char. slope
  Vc : real := 0.0 -- centre frequency input voltage
);
  port(terminal vol_in,vco_op: electrical);
end entity VCO;
architecture conceptual of VCO is
  CONSTANT TwoPi: real := 6.283118530718; -- 2pi
  quantity v_in across i_in through vol_in to electrical'reference;
  quantity v_op across i_op through vco_op to electrical'reference;
begin
  V_op == (amp*sin((2.0*22.0/7.0*(fc+(df*(v_in-Vc))))*real(time' pos(now))*1.0e-15));
end architecture conceptual;
```

### A.9 Oscillator

```plaintext
PACKAGE electricalsystem IS
  NATURE electrical IS real ACROSS real THROUGH;
  FUNCTION SIN(X:real) RETURN real;
  FUNCTION COS(X:real) RETURN real;
  FUNCTION EXP(X:real) RETURN real;
END PACKAGE electricalsystem;

-- Insert code for res(A.19),cap(A.20),diode(A.18),NPN_trn(A.16.2)and
-- PNP_trn(A.16.3) and op-amp(A.12.2)

--> Constant Voltage source
-----------------------------
use work.electricalsystem.all;
ENTITY voltSource IS
generic(amp:real:=22.0);
  PORT( TERMINAL ta2,tb2 : electrical);
END voltSource;
ARCHITECTURE voltbehavior OF voltSource IS
```
terminal t1: electrical;
quantity V_volt across i_volt through t1 to tb2;
quantity V_drop across i_drop through ta2 to t1;

BEGIN
  V_volt == amp;
  V_drop == i_drop*100.0;
END ARCHITECTURE voltbehavior;

--------------- WEIN BRIDGE OSCILLATOR --------------------
use work.electricalsystem.all;
entity wein_bridge_osc is
  port( terminal signal_out :electrical);
end entity wein_bridge_osc;
architecture struct of wein_bridge_osc is
  --> components
  component op_amp is
    port(terminal inverting_ip,non_inverting_ip,output :electrical);
  end component;
  for all:op_amp use entity work.op_amp(struct);

  component diode
    generic (Isat : real := 1.0e-14; -- saturation current
               n : real := 1.0; -- emmission coefficient
               bv : real := 1.0; -- reverse breakdown voltage
               ibv : real := 1.0e-3; -- Breakdown current
               rds : real := 1.0 -- Ohmic resistamce);
    port (terminal pos, neg : electrical);
  end component;
  for all : diode use entity work.Diode(behav);

  component capacitor is
    generic(cap :real:=1.0);
    port(terminal c_in,c_out: electrical);
  end component;
  for all: capacitor use entity work.capacitor(behav);

  component resistor is
    generic(res :real:=1.0 );
    port(terminal r_in,r_out: electrical);
  end component;
  for all: resistor use entity work.resistor(behav);

  terminal t1,t2,t3,t4: electrical;

begin
  op_amplifier : op_amp port map(inverting_ip=>t4,
                                  non_inverting_ip=>t1,output=>t3);
  D1 : diode port map(t3,signal_out);
  D2 : diode port map(signal_out,t3);
  R1_a : resistor generic map(18.0e3)
         port map(t4,electrical'reference);
  R1_b : resistor generic map(32.0e3)
A.10  **FM-to-AM Converter**

```vhdl
PACKAGE electricalSystem IS
    NATURE electrical IS real ACROSS real THROUGH;
    FUNCTION SIN (X : real ) RETURN real;
    FUNCTION COS (X : real ) RETURN real;
    FUNCTION EXP (X : real ) RETURN real;
    FUNCTION SQRT (X : real) RETURN real;
END PACKAGE electricalSystem;

---------------------- FM to AM Converter ---------------------
USE work.electricalSystem.all;

ENTITY FM_2_AM_Converter IS
    generic (freq_fm : real := 1.0);
    port (terminal Signal_in, Signal_out : electrical);
END FM_2_AM_Converter;

ARCHITECTURE behav OF FM_2_AM_Converter IS
    CONSTANT k : real := 0.4;
    CONSTANT lp : real := 1.0e-3;
    CONSTANT ls : real := 1.0e-3;
    CONSTANT rp : real := 10.0;
    CONSTANT rs : real := 10.0;

    --> Q = 2*PI*Freq*L/R : for 10.7 MHz -> q=6723
    terminal temp1, temp2: electrical;

    quantity v rp across i rp through Signal_in to temp1;
    quantity v rs across i rs through temp2 to Signal_out;

    quantity V cp across i cp through Signal_in to electrical'reference;
    quantity V cs across i cs through Signal_out to electrical'reference;

    QUANTITY V lp ACROSS i lp Through temp1 to electrical'reference;
    quantity v ls across i ls through temp2 to electrical'reference;

    quantity m : real; -- mutual inductance;

BEGIN -- behavior
```

```vhdl
end struct;
```
brk : break i_lp => 0.0, i_ls => 0.0, v_cp => 0.0, v_cs => 0.0;

mutual : m = k * sqrt(lp*ls);
voltp : v_lp = lp * i_lp'do t + m * i_ls'dot;
volts : v_ls = ls * i_ls'dot + m * i_lp'dot;

i_cp = (25.331/(freq_fm*freq_fm))*v_cp'dot;
i_cs = (25.331/(freq_fm*freq_fm))*v_cs'dot;

v_rp = rp * i_rp;
v_rs = rs * i_rs;

END behav;

A.11 Transformers

A.11.1 Untuned Transformer

PACKAGE electricalSystem IS
  NATURE electrical IS real ACROSS real THROUGH;
  FUNCTION SIN (X : real ) RETURN real;
  FUNCTION COS (X : real ) RETURN real;
  FUNCTION EXP (X : real ) RETURN real;
  FUNCTION SQRT (X : real) RETURN real;
END PACKAGE electricalSystem;

---------------------- Transformer -------------------------------
USE work.electricalSystem.all;
ENTITY TRANSFORMER IS
generic (
  k : real := 0.9999; -- coupling factor
  lp : real := 10.0e-3; -- primary inductance
  ls : real := 10.0e-3); -- secondary inductance
  port (terminal t1, t2, t3, t4 : electrical);
END TRANSFORMER;

ARCHITECTURE behavior OF TRANSFORMER IS
QUANTITY Vlp ACROSS ilp Through t1 to t2;
quantity vls across ils through t3 to t4;
quantity m : real ; -- mutual inductance;

BEGIN -- behavior
  brk : break ilp => 0.0, ils => 0.0;
  mutual : m = k * sqrt(lp*ls);
  voltp : v_lp = lp * i_lp'dot + m * i_ls'dot;
  volts : v_ls = ls * i_ls'dot + m * i_lp'dot;
END behavior

A.11.2 Untuned Secondary Tapped Transformer

PACKAGE electricalsystem IS
  NATURE electrical IS real ACROSS real THROUGH;
  FUNCTION SIN(X:real) RETURN real;
  FUNCTION EXP(X:real) RETURN real;
FUNCTION SQRT(X:real) RETURN real;
END PACKAGE electricalsystem;
---------------------------> transformer <-----------------------------
USE work.electricalSystem.all;

ENTITY TRANSFORMER IS
generic (
  k1 : real := 0.9999; -- coupling factor
  k2 : real := 0.9999; -- coupling factor
  k3 : real := 0.9999; -- coupling factor
  lp : real := 10.0e-3; -- primary inductance
  ls : real := 10.0e-3; -- secondary inductance
  tap : real := 0.5); -- tapping point
port (terminal v_in,gnd_in,t_up,t_mid,t_down : electrical);
END TRANSFORMER;

architecture trans_behav of transformer is

terminal t1,t2,t3,t4:electrical;
quantity m1,m2,m3:real;
quantity vin_1 across iin_1 through t1 to gnd_in;
quantity vout_1 across iout_1 through t2 to t3;
quantity vout_2 across iout_2 through t3 to t4;
quantity vr1 across ir1 through v_in to t1;
quantity vr2 across ir2 through t2 to t_up;
quantity vr3 across ir3 through t3 to t_mid;
quantity vr4 across ir4 through t4 to t_down;

begin
  brk : break iin_1=>0.0, iout_2=>0.0, iout_1=>0.0;

  m1==k1*sqrt(lp*ls*(1.0-tap));
  m2==k2*sqrt(lp*ls*tap);
  m3==k3*sqrt(ls*tap*ls*(1.0-tap));

  vr1 ==ir1*1.0;
  vr2 ==ir2*1.0;
  vr3 ==ir3*1.0;
  vr4 ==ir4*1.0;

  vin_1 == lp*iin_1'dot + m2*iout_2'dot+ m1*iout_1'dot;
  vout_1 == ls*(1.0-tap)*iout_1'dot+ m1*iin_1'dot + m3*iout_2'dot;
  vout_2 == ls*tap*iout_2'dot + m2*iin_1'dot + m3*iout_1'dot;

end architecture trans_behav;

A.11.3  Double Tuned Transformer

PACKAGE electricalSystem IS
  NATURE electrical IS real ACROSS real THROUGH;
  FUNCTION SIN (X : real ) RETURN real;
  FUNCTION COS (X : real ) RETURN real;
  FUNCTION EXP (X : real ) RETURN real;
  FUNCTION SQRT (X : real) RETURN real;
END PACKAGE electricalSystem;
ENTITY tuned_transformer IS
    generic (freq_fm : real := 1.0);
    port (terminal Signal_in, Signal_out : electrical);
END tuned_transformer;

ARCHITECTURE behav OF tuned_transformer IS
    CONSTANT k :real:=0.4;
    CONSTANT lp :real:=1.0e-3;
    CONSTANT ls :real:=1.0e-3;
    CONSTANT rp :real:=10.0;
    CONSTANT rs :real:=10.0;

    --> Q = 2*PI*Freq*L/R : for 10.7 MHz -> q=6723

    terminal temp1,temp2: electrical;
    quantity v_rp across i_rp through Signal_in to temp1;
    quantity v_rs across i_rs through temp2 to Signal_out;

    quantity V_cp across i_cp through Signal_in to electrical'reference;
    quantity V_cs across i_cs through Signal_out to electrical'reference;

    QUANTITY V_lp ACROSS i_lp Through temp1 to electrical'reference;
    quantity v_ls across i_ls through temp2 to electrical'reference;

    quantity m : real ; -- mutual inductance;

BEGIN -- behavior
    brk : break i_lp => 0.0, i_ls => 0.0,v_cp=>0.0,v_cs=>0.0;

    mutual : m = k * sqrt(lp*ls);
    voltp : v_lp == lp * i_lp'dot + m * i_ls'dot;
    volts : v_ls == ls * i_ls'dot + m * i_lp'dot;

    i_cp == (25.331/(freq_fm*freq_fm))*v_cp'dot;
    i_cs == (25.331/(freq_fm*freq_fm))*v_cs'dot;

    -- modify this if u want to use another of Lp and Ls
    -- c =1/(2*PI*F)*(2*PI*F)*L
    -- cal. using the value of Inductance as 1.0mH

    v_rp == rp *i_rp;
    v_rs == rs *i_rs;

END behav;

A.11.4 Double Tuned Secondary Tapped Transformer

PACKAGE electricalSystem IS
    NATURE electrical IS real ACROSS real THROUGH;
    FUNCTION SIN (X : real ) RETURN real;
    FUNCTION COS (X : real ) RETURN real;
    FUNCTION EXP (X : real ) RETURN real;

FUNCTION SQRT (X : real) RETURN real;
END PACKAGE electricalSystem;

----------------- TUNED TRANSFORMER ---------------------------
USE work.electricalSystem.all;

ENTITY tuned_tapped IS
  generic (freq_upsec: real := 1.0; freq_downsec: real:=1.0);
  port (terminal Signal_in, in_gnd, T_up,T_mid,T_down: electrical);
END tuned_tapped;

ARCHITECTURE behav OF tuned_tapped IS

  CONSTANT k : real := 0.2;
  CONSTANT lp : real := 1.0e-3;
  CONSTANT ls : real := 1.0e-3; -- Ls1 == Ls2 = Ls=1.0e-3
  CONSTANT rp : real := 1.256e3; -- Rs1==Rs2 =10.0
  CONSTANT rs : real := 10.0; -- Rs1==Rs2 =10.0

  --> Q = 2*PI*Freq*L/R : for 10.7 MHz -> q=6723

  terminal t1,t2,t3: electrical;

  quantity v_rp across i_rp through Signal_in to t1;
  quantity v_rs1 across i_rs1 through t2 to T_up;
  quantity v_rs2 across i_rs2 through t_mid to T_down;
  quantity V_cp across i_cp through Signal_in to in_gnd;
  quantity V_cs1 across i_cs1 through t_up to t_mid;
  quantity V_cs2 across i_cs2 through t_down to t_mid;

  QUANTITY V_lp ACROSS i_lp Through t1 to in_gnd;
  quantity v_ls1 across i_ls1 through t2 to t_mid;
  quantity v_ls2 across i_ls2 through t_down to t3;

  quantity m : real; -- mutual inductance;

BEGIN -- behavior

  brk : break i_lp => 0.0, i_ls1 => 0.0,i_ls2=>0.0,v_cp=>0.0,
       v_cs1=>0.0,v_cs2=>0.0;

  mutual : m = k * sqrt(lp*ls);

  voltp : v_lp == lp * i_lp'dot + m * i_ls1'dot;
  voltst1 : v_ls1 == ls * i_ls1'dot + m * i_lp'dot;
  voltst2 : v_ls2 == ls * i_ls2'dot + m * i_lp'dot;

  i_cp = 0.22125e-12 * v_cp'dot; -- cal. using the value of Ind
  i_cs1 = (25.331/(freq_upsec*freq_upsec))* v_cs1'dot; -- as 1mH
  i_cs2 = (25.331/(freq_downsec*freq_downsec))* v_cs2'dot;

  v_rp == rp * i_rp;
  v_rs1 == rs * i_rs1;
  v_rs2 == rs * i_rs2;

END behav;

A.12 Operational Amplifier
A.12.1 Simple Operational Amplifier

PACKAGE electricalsystem IS
NATURE electrical IS real ACROSS real THROUGH;
FUNCTION SIN(X:real) RETURN real;
FUNCTION EXP(X:real) RETURN real;
END PACKAGE electricalsystem;

--------------- OP AMP -----------------------------
use work.electricalsystem.all;

entity op_amp is
port(terminal inverting_ip,non_inverting_ip,output :electrical);
end entity op_amp;

architecture struct of op_amp is

Constant R_in:real:=1.0e6;
Constant R_out:real:=1.0;

terminal t1:electrical;

quantity v_in across i_in through non_inverting_ip to inverting_ip;
quantity v_gain across i_gain through t1 to electrical'reference;
quantity v_drop across i_drop through t1 to output;

BEGIN

V_in==i_in*R_in;
V_gain==V_in*(1000.0);
V_drop==i_drop*R_out;

end architecture struct;

A.12.2 Simple Operational Amplifier(Using 5 BJT’s)

PACKAGE electricalsystem IS
NATURE electrical IS real ACROSS real THROUGH;
FUNCTION SIN(X:real) RETURN real;
FUNCTION EXP(X:real) RETURN real;
END PACKAGE electricalsystem;

-- Include code for NPN_trans(A.16.2), PNP_tran(A.16.3), res(A.19) &
-- cap(A.20)
-- ********* Structural Model Of a simple High Frequency OpAmp ********--
use work.electricalsystem.all;

entity op_amp is
port(terminal inverting_ip,non_inverting_ip,output :electrical);
end entity op_amp;

architecture struct of op_amp is

--> components

COMPONENT trans_pnp is
port( terminal emitter,base,collector : electrical);
end component;
for all : trans_pnp use entity work.trans_pnp(trans_behav);

COMPONENT trans_npn is
port( terminal emitter,base,collector : electrical);
end component;
for all : trans_npn use entity work.trans_npn(trans_behav);

component resistor is
generic(res :real:=1.0);
port(terminal r_in,r_out: electrical);
end component;
for all: resistor use entity work.resistor(behav);

component voltsource is
generic(amp:real:=22.0);
PORT( TERMINAL ta2,tb2 : electrical);
end component;
for all: voltsource use entity work.voltsource(voltbehavior);

terminal t1,t2,t3,t4,t5,t6,t7,t8,t9,t10:electrical;
terminal V_pos,V_neg: electrical;
BEGIN

Q01_npn: trans_npn port map(emitter=>T2 ,base=>T1,collector=>T9);
Q02_npn: trans_npn port map(emitter=>T2 ,base=>T3,collector=>T4);
Q03_npn: trans_npn port map(emitter=>T5 ,base=>T6,collector=>T2);
Q04_npn: trans_pnp port map(emitter=>T7 ,base=>T4,collector=>T8);
Q05_npn: trans_npn port map(emitter=>output ,base=>T8,collector=>V_pos);

Res_i1 : resistor generic map(1.0e3) port map(inverting_ip,T1);
Res_i2 : resistor generic map(1.0e3) port map(non_inverting_ip,T3);
Res_a : resistor generic map(220.0e3) port map(T6,V_pos);
Res_c1 : resistor generic map(13.0e3) port map(T9,V_pos);
Res_c2 : resistor generic map(13.0e3) port map(V_pos,T4);
Res_e4 : resistor generic map(10.0e3) port map(V_pos,T7);
Res_b : resistor generic map(20.0e3) port map(T6,V_neg);
Res_e3 : resistor generic map(1.3e3) port map(T5,V_neg);
Res_c4 : resistor generic map(21.0e3) port map(T8,V_neg);
Res_e5 : resistor generic map(12.0e3) port map(output,V_neg);

vpos : voltsource generic map(amp=>15.0) -- test case
port map(V_pos,electrical'reference);
vneg : voltsource generic map(amp=>-15.0) -- test case
port map(V_neg,electrical'reference);

end architecture struct;
A.12.3 Operational Amplifier (L741)

PACKAGE electricalsystem IS
  NATURE electrical IS real ACROSS real THROUGH;
  FUNCTION SIN(X:real) RETURN real;
  FUNCTION EXP (X:real) RETURN real;
END PACKAGE electricalsystem;

-- Include code for NPN_trans(A.16.2), PNP_tran(A.16.3), res(A.19) &
-- cap(A.20)
-- ******** Structural Model Of INPUT SECTION OF OpAmp LM701 **********--
use work.electricalsystem.all;

entity op_amp_ip is
  port(terminal inverting_ip,non_inverting_ip,out1,out2,V_pos,V_neg :
       electrical);
end entity op_amp_ip;

architecture struct of op_amp_ip is

  --> components

  COMPONENT trans_pnp is
    port(terminal emitter,base,collector : electrical);
  end component;
  for all : trans_pnp use entity work.trans_pnp(trans_behav);

  COMPONENT trans_npn is
    port(terminal emitter,base,collector : electrical);
  end component;
  for all : trans_npn use entity work.trans_npn(trans_behav);

  component resistor is
    generic(res :real:=1.0);
    port(terminal r_in,r_out: electrical);
  end component;
  for all: resistor use entity work.resistor(behav);

begin

Q01_npn: trans_npn port map(emitter=>T1,base=>
non_inverting_ip,collector=>T2);
Q02_npn: trans_npn port map(emitter=>T3,base=>inverting_ip
,collector=>T2);
Q03_pnp: trans_pnp port map(emitter=>T1,base=>T6
,collector=>T5);
Q04_pnp: trans_pnp port map(emitter=>T3,base=>T6
,collector=>out2);
Q05_npn: trans_npn port map(emitter=>T7,base=>T8
,collector=>T5);
Q06_npn: trans_npn port map(emitter=>T9,base=>T8
,collector=>out2);
Q07_npn: trans_npn port map(emitter=>T8,base=>T5
,collector=>V_pos);

end

Q08_pnp: trans_pnp port map(emitter=>V_pos ,base=>T2 ,
collector=>T2);
Q09_pnp: trans_pnp port map(emitter=>V_pos ,base=>T2 ,
collector=>T6);
Q10_npn: trans_npn port map(emitter=>T11 ,base=>T10 ,
collector=>T6);
Q11_npn: trans_npn port map(emitter=>V_neg ,base=>T10 ,
collector=>T10);
Q12_pnp: trans_pnp port map(emitter=>V_pos ,base=>out1 ,
collector=>out1);

Res_1 : resistor generic map(1.0e3)
port map(T7,V_neg);
Res_2 : resistor generic map(1.0e3)
port map(T9,V_neg);
Res_3 : resistor generic map(50.0e3)
port map(T8,V_neg);
Res_4 : resistor generic map(5.0e3)
port map(T11,V_neg);
Res_5 : resistor generic map(30.0e3)
port map(out1,T10);

end architecture struct;

-- ** Structural Model Of BUFFER SECTION OF OpAmp LM701 *******--

use work.electricalsystem.all;

entity op_amp_mid is
port(terminal out1,out2,out3,out4,V_pos,V_neg :electrical);
end entity op_amp_mid;

architecture struct of op_amp_mid is

--> components

COMPONENT trans_pnp is
port( terminal emitter,base,collector : electrical);
end component;
for all : trans_pnp use entity work.trans_pnp(trans_behav);

COMPONENT trans_npn is
port( terminal emitter,base,collector : electrical);
end component;
for all : trans_npn use entity work.trans_npn(trans_behav);

component resistor is
generic(res :real:=1.0 );
port(terminal r_in,r_out: electrical);
end component;
for all: resistor use entity work.resistor(behav);

terminal t1,t2,t3,t4:electrical;

BEGIN
Q13_pnp: trans_pnp port map(emitter=>V_pos, base=>out1, collector=>out3);
Q14_pnp: trans_npn port map(emitter=>out4 ,base=>t1 , collector=>out3);
Q15_npn: trans_npn port map(emitter=>T3, base=>out2, collector=>out4);
Q17_npn: trans_npn port map(emitter=>T4, base=>T3, collector=>out4);

res_q16 : resistor generic map(5.0)
port map(V_neg,out2);
Res_7 : resistor generic map(4.5e3)
port map(T1,out3); Res_8 : resistor generic map(7.5e3)
port map(T1,out4); Res_9 : resistor generic map(50.0e3)
port map(T3,V_neg); Res_10 : resistor generic map(50.0)
port map(T4,V_neg);

end architecture struct;

-- ****** Structural Model Of OUTPUT SECTION of OpAmp ******************--

use work.electricalsystem.all;

entity op_amp_op is
    port(terminal out3,out4,output,V_pos,V_neg :electrical);
end entity op_amp_op;

architecture struct of op_amp_op is

--> components

COMPONENT trans_pnp is
    port( terminal emitter,base,collector : electrical);
end component;
for all : trans_pnp use entity work.trans_pnp(trans_behav);

COMPONENT trans_npn is
    port( terminal emitter,base,collector : electrical);
end component;
for all : trans_npn use entity work.trans_npn(trans_behav);

component resistor is
    generic(res :real:=1.0 );
    port(terminal r_in,r_out: electrical);
end component;
for all: resistor use entity work.resistor(behav);

terminal t1,t2,t3,t4: electrical;

BEGIN

Q14_npn: trans_npn port map(emitter=>t1, base=>out3, collector=>V_pos);
Q15_npn: trans_npn port map(emitter=>output,base=>T1, collector=>out3);
Q20_pnp: trans_pnp port map(emitter=>t2, base=>out4, collector=>V_neg);

res6: resistor generic map(25.0)
port map(T1,output);
res7: resistor generic map(50.0)
port map(output,T2);
end architecture struct;

--------------------------------------- OP_AMP ----------------------------------------

use work.electricalsystem.all;
entity op_amp is
  port(terminal pos_ip,neg_ip,op,V_pos,V_neg: electrical);
end entity op_amp;

architecture struct of op_amp is
  --> components
  component op_amp_ip is
    port(terminal inverting_ip,non_inverting_ip,out1,out2,V_pos,V_neg :electrical);
  end component;
  for all:op_amp_ip use entity work.op_amp_ip(struct);

  component op_amp_mid is
    port(terminal out1,out2,out3,out4,V_pos,V_neg :electrical);
  end component;
  for all:op_amp_mid use entity work.op_amp_mid(struct);

  component op_amp_op is
    port(terminal out3,out4,output,V_pos,V_neg :electrical);
  end component;
  for all:op_amp_op use entity work.op_amp_op(struct);

  terminal t1,t2,t3,t4,t5: electrical;
begin
  op_ip : op_amp_ip port map(neg_ip,pos_ip,t2,t3,V_pos,V_neg);
  op_mp : op_amp_mid port map(t2,t3,t4,t5,V_pos,V_neg);
  op_op : op_amp_op port map(t4,t5,op,V_pos,V_neg);
end architecture struct;

A.13  Crystal

--PACKAGE electricalsystem IS
  NATURE electrical IS real ACROSS real THROUGH;
  FUNCTION SIN(X:real) RETURN real;
  FUNCTION EXP(X:real) RETURN real;
END PACKAGE electricalsystem;

--Include code for inductor(A.21),res(A.19) and cap(A.20)
---------------------- CRYSTAL ----------------------
use work.electricalsystem.all;

entity crystal is
  generic (freq: real:=1.0);

architecture struct of crystal is

--> Components
  component capacitor is
    generic(cap :real:=1.0);
    port(terminal c_in,c_out: electrical);
    end component;
  for all: capacitor use entity work.capacitor(behav);

  component inductor is
    generic(ind :real:=1.0 );
    port(terminal r_in,r_out: electrical);
    end component;
  for all: inductor use entity work.inductor(behav);

  component resistor is
    generic(res :real:=1.0 );
    port(terminal r_in,r_out: electrical);
    end component;
  for all: resistor use entity work.resistor(behav);

  terminal T1,t2: electrical;

CONSTANT Q : real:=10000.0; -- CONSTANTS
CONSTANT R : real:=600.0; -- FOR THE
CONSTANT Cs : real:=8.0e-12; -- CRYSTAL.

BEGIN

  Res : resistor generic map(R)
      port map(t2,B);
  C_s : capacitor generic map(Cs)
      port map(A,B);
  Ind : inductor generic map(Q*R/freq)
      port map(A,t1);
  C_p : capacitor generic map(1.0/((Q*R/freq)*(freq-
                                          1.0/(2.0*R*Q*Cs)))))
      port map(t1,t2);

end struct;

A.14 Frequency Doubler

PACKAGE electricalsystem IS
  NATURE electrical IS real ACROSS real THROUGH;
  FUNCTION SIN(X:real) RETURN real;
  FUNCTION EXP(X:real) RETURN real;
END PACKAGE electricalsystem;

-- Insert code for simple op-amp(A.12.1) and mixer(A.3.1)
-------------------------- PHASE SHIFTER --------------------------
use work.electricalsystem.all;

entity phase_shifter is
  generic(freq:real:=1.0);
  port(terminal sig_in,sig_out: electrical);
end entity phase_shifter;
architecture struct of phase_shifter is

--> Component
    component op_amp is
        port(terminal inverting_ip,non_inverting_ip,output :electrical);
    end component;
    for all: op_amp use entity work.op_amp(struct);

    terminal t1,t2,t3,t4,t5 :electrical;
    quantity v_r1 across i_r1 through Sig_in to t1;
    quantity v_r2 across i_r2 through t1 to sig_out;
    quantity v_r3 across i_r3 through sig_in to t2;
    quantity v_c1 across i_c1 through t2 to electrical'reference;

BEGIN
    op_1 : op_amp port map(T1,T2,sig_out);
    v_r1==i_r1*10.0e3;
    v_r2==i_r2*10.0e3;
    v_r3==i_r3*10.0e3;
    i_c1==(1.591549430e-5/freq)*v_c1'dot;

end architecture struct;

----------------------------- FREQUENCY DOUBLER -----------------------------
use work.electricalsystem.all;

entity freq_doubler is
    generic(freq:real:=1.0);
    port (terminal signal_in,signal_out:electrical);
end entity freq_doubler;

architecture struct of freq_doubler is

--> Components
    component mixer_conceptual is
        port(terminal rf_in,rf_gnd,lo_osc,lo_gnd,mixer_op,op_gnd :electrical);
    end component;
    for all: mixer_conceptual use entity work.mixer_conceptual (mixer_conceptual_behav);

    component phase_shifter is
        generic(freq:real:=1.0);
        port(terminal sig_in,sig_out: electrical);
    end component;
    for all: phase_shifter use entity work.phase_shifter(struct);

    terminal t1: electrical;

begin
    freq_d : phase_shifter generic map(freq)
        port map(signal_in,t1);
    mixer : mixer_conceptual port map(signal_in,
        electrical'reference,t1,
        electrical'reference,sig_out,electrical'reference);

end struct;
A.15 AM Detector

--- BLOCK DIAGRAM ---

--- o____| Rectifier|____| lowpass |____| dc_block|____| lowpass|____o

---

-- PACKAGE electricalsystem IS
  NATURE electrical IS real ACROSS real THROUGH;
  FUNCTION SIN(X:real) RETURN real;
  FUNCTION COS(X:real) RETURN real;
  FUNCTION EXP(X:real) RETURN real;
END PACKAGE electricalsystem;

--Include code for res(A.19),cap(A.20),op-amp(A.12.1),diode(A.18)
-- LPF(A.7.1)

------------------------- low pass buffer ------------------
use work.electricalsystem.all;

entity low_pass_buffer is
  port (terminal v_in, v_out : electrical);
end entity low_pass_buffer;

architecture struct of low_pass_buffer is
  -- components
  component op_amps is
    port (terminal inverting_ip, non_inverting_ip, output : electrical);
  end component;
  for all:op_amps use entity work.op_amps(struct);
  
  component resistor is
    generic(res:real:=1.0);
    PORT( TERMINAL ta2, tb2 : electrical);
  end component;
  for all:resistor use entity work.op_amps(struct);
  
  component capacitor is
    generic(cap:real:=1.0);
    PORT( TERMINAL ta2, tb2 : electrical);
  end component;
  for all:capacitor use entity work.op_amps(struct);

  terminal T1, T2, T3: electrical;
BEGIN

  R1: resistor generic map(2.2e3)
      port map(V_in, T1);
  R2: resistor generic map(10.0e3)
      port map(T1, V_out);
  C1: capacitor generic map(0.0047e-6)
      port map(T1, V_out);

END
op: op_amp port map(inverting_ip=>T1,non_inverting_ip=>electrical'reference,output=>V_out);
end struct;

------------------------- SIMPLE DIODE RECTIFIER ------------------
use work.electricalsystem.all;

entity super_diode_rectifier is
  port (terminal v_in, v_out : electrical);
end entity super_diode_rectifier;

architecture struct of super_diode_rectifier is

  --> components

  component diode
generic (    
    Isat : real := 1.0e-14; -- saturation current    
    n : real := 1.0; -- emmission coefficient    
    bv : real := 15.0; -- reverse breakdown voltage    
    ibv : real := 1.0e-3; -- Breakdown current    
    rds : real := 1.0 -- Ohmic resistance
  );
  port (terminal pos, neg : electrical);
end component;
for all : diode use entity work.Diode(behav);

  component resistor is
generic(res:real:=1.0);
  PORT( TERMINAL ta2,tb2 : electrical);
end component;
for all: resistor use entity work.resistor(behav);

BEGIN

  D1: diode port map(V_in,V_out);
  R1: resistor generic map(10.0e3) port map(V_out,electrical'reference);

end struct;

------------------------- DC BLOCKING BUFFER ------------------
use work.electricalsystem.all;

entity dc_blocking_buffer is
  port (terminal v_in, v_out : electrical);
end entity dc_blocking_buffer;

architecture struct of dc_blocking_buffer is

  --> components

  component op_amp is
    port(terminal inverting_ip,non_inverting_ip, output : electrical);
  end component;
for all:op_amp use entity work.op_amp(struct);

  component resistor is
    generic(res:real:=1.0);
    PORT( TERMINAL ta2,tb2 : electrical);
end component;
for all: resistor use entity work.resistor(behav);

component capacitor is
  generic(cap:real:=1.0);
  PORT( TERMINAL ta2, tb2 : electrical);
end component;
for all: capacitor use entity work.capacitor(behav);

terminal T1, T2, T3: electrical;

BEGIN

R1: resistor generic map(10.0e3)
  port map(T1, electrical'reference);
R2: resistor generic map(10.0e3)
  port map(T2, V_out);
C1: capacitor generic map(0.1e-6)
  port map(V_in, T1);

op: op_amp port map(inverting_ip=>T2, non_inverting_ip=>T1,
  output=>V_out);

end struct;

------------------------ AM DEMODULATOR ------------------------
use work.electricalsystem.all;
entity am_demodulator is
  port (terminal signal_in, signal_out: electrical);
end entity am_demodulator;
architecture struct of am_demodulator is

--> components
component super_diode_rectifier is
  port (terminal v_in, v_out : electrical);
end component;
for all: super_diode_rectifier use entity
  work.super_diode_rectifier(struct);

cOMPONENT low_pass_buffer IS
  port (terminal v_in, v_out : electrical);
end component;
for all: low_pass_buffer use entity work.low_pass_buffer(struct);

cOMPONENT dc_blocking_buffer IS
  port (terminal v_in, v_out : electrical);
end component;
for all: dc_blocking_buffer use entity work.dc_blocking_buffer
  (struct);

cOMPONENT low_pass_filter IS
  generic(freq:real:=100.0e6);
  port (terminal V_in, V_out : electrical);
end component;
for all: low_pass_filter use entity work.low_pass_filter(struct);

terminal T4, T2, T3: electrical;

BEGIN

simplerectifier : super_diode_rectifier port map(Signal_in, T2);
lowpassbuffer : low_pass_buffer port map(T2, T3);
A.16 Transistors

A.16.1 Trans_T_Equivalent Structural Model

PACKAGE electricalsystem IS
    NATURE electrical IS real ACROSS real THROUGH;
    FUNCTION SIN(X:real) RETURN real;
    FUNCTION EXP(X:real) RETURN real;
END PACKAGE electricalsystem;

--Include code for res(A.19),cap(A.20) and Ind(A.21)

--------------------VOLTAGE DEPT. CURRENT SOURCE---------------------
use work.electricalsystem.all;

entity current_source is
    generic(gain :real:=0.88 );
    port(terminal i_in,i_out,v_sample_in,v_sample_out: electrical);
end entity current_source;

architecture behav of current_source is
    terminal T1,t2: electrical;
    quantity v_sample across i_sample through v_sample_in to v_sample_out;
    quantity i_gen through i_in to t1;
    quantity vr across ir through t1 to t2;
    quantity vl across il through t2 to i_out;
begin
    v_sample==i_sample*1.0e-20; -- LOW RESISTANCE IN SERIES
    i_gen==i_sample*gain;
    vr == ir*0.999959619;
    vl ==1.431091826e-11*il'dot;
    --- angle = .2628+ATAN(freq/22.7e9)
    --- res = cos(angle)
    --- ind = sin(angle)/(2*PI*freq)
end architecture behav;

--################## Structural Model of a RF bjt ########################
use work.electricalsystem.all;

entity trans_t_eq_struct is
    port(terminal Emitter,Base,Collector : electrical);
end trans_t_eq_struct;

architecture trans_behav of trans_t_eq_struct is
    terminal t1,t2,t3,t4,t5,t6,t7 :electrical;
    --> Components are defined here
A.16.2 Behavioral Model of NPN_Transistor

PACKAGE electricalSystem IS
  NATURE electrical IS real ACROSS real THROUGH;
  FUNCTION POW(X,Y: real) RETURN real;
end
function pow (x :integer; y : real) return real;
FUNCTION SIN(X : real) RETURN real;
FUNCTION EXP(X : real) RETURN real;
FUNCTION SQRT(X : real) RETURN real;
END PACKAGE electricalSystem;

use work.electricalsystem.all;

entity trans_t_eq is
port( terminal emitter,base,collector : electrical);
end trans_t_eq;

architecture trans_behav of trans_t_eq is

terminal t1,t2,t3,t4,t5,e,b :electrical;
constant Lb :real:=0.5e-9;
constant rb1 :real:=1.0;
constant rb2 :real:=3.1;
constant rb3 :real:=2.7;
constant r_pi :real:=110.0;
constant c_pi :real:=18.0e-12;
constant gm :real:=0.88;
constant cc1 :real:=0.091e-12;
constant cc2 :real:=0.048e-12;
constant cc3 :real:=0.023e-12;
constant Le :real:=0.2e-9;
constant Rbase:real:=22.0;
constant Remit:real:=0.6;

quantity v1 across i1 through b to t1;
quantity v2 across i2 through t1 to t2;
quantity v3 across i3 through t2 to t3;
quantity v4 across i4 through t3 to t4;
quantity v_pi across i5 through t4 to t5;
quantity i6 through t4 to t5;
quantity v7 across i7 through t1 to collector;
quantity v8 across i8 through t2 to collector;
quantity v9 across i9 through t3 to collector;
quantity v10 across i10 through t5 to e;
quantity v11 across i11 through collector to t5;
quantity v_base across i_base through base to b;
quantity v_emit across i_emit through e to emitter;

BEGIN
v1 ==Lb*i1'dot;
v2 ==i2*rb1;
v3 ==i3*rb2;
v4 ==i4*rb3;
v_pi==i5*r_pi;
i6 ==c_pi*v_pi'dot;
i7 ==cc1*v7'dot;
i8 ==cc2*v8'dot;
i9 ==cc3*v9'dot;
v10 ==Le*i10'dot;
i11 ==gm*v_pi;
v_base==rbase*i_base;
v_emit==remit*i_emit;
end architecture trans_behav;
A.16.3 Behavioral Model of PNP_Transistor

PACKAGE electricalsystem IS
  NATURE electrical IS real ACROSS real THROUGH;
  FUNCTION SIN(X:real) RETURN real;
  FUNCTION EXP(X:real) RETURN real;
END PACKAGE electricalsystem;

use work.electricalsystem.all;

entity trans_t_eq is
  port( terminal emitter,base,collector : electrical);
end trans_t_eq;

architecture trans_behav of trans_t_eq is

  terminal t1,t2,t3,t4,t5,e,b :electrical;
  constant Lb :real:=0.5e-9;
  constant rb1 :real:=1.0;
  constant rb2 :real:=3.1;
  constant rb3 :real:=2.7;
  constant r_pi :real:=110.0;
  constant c_pi :real:=18.0e-12;
  constant gm :real:=0.88;
  constant cc1 :real:=0.091e-12;
  constant cc2 :real:=0.048e-12;
  constant cc3 :real:=0.023e-12;
  constant Le :real:=0.2e-9;
  constant Rbase:real:=22.0;
  constant Remit:real:=0.6;

  quantity v1 across i1 through t1 to b;
  quantity v2 across i2 through t2 to t1;
  quantity v3 across i3 through t3 to t2;
  quantity v4 across i4 through t4 to t3;
  quantity v_pi across i5 through t5 to t4;
  quantity i6 through t5 to t4;
  quantity v7 across i7 through collector to t1;
  quantity v8 across i8 through collector to t2;
  quantity v9 across i9 through collector to t3;
  quantity v10 across i10 through e to t5;
  quantity v11 across i11 through t5 to collector ;
  quantity v_base across i_base through b to base;
  quantity v_emit across i_emit through emitter to e;

BEGIN
  v1 ==Lb*i1'dot;
  v2 ==i2*rb1;
  v3 ==i3*rb2;
  v4 ==i4*rb3;
  v_pi==i5*r_pi;
  i6 ==c_pi*v_pi'dot;
  i7 ==cc1*v7'dot;
  i8 ==cc2*v8'dot;
  i9 ==cc3*v9'dot;
  v10 ==Le*i10'dot;
  i11 ==gm*v_pi;
  v_base==rbase*i_base;
A.17 Structural Model of the FET

PACKAGE electricalsystem IS
   NATURE electrical IS real ACROSS real THROUGH;
   FUNCTION SIN(X:real) RETURN real;
   FUNCTION EXP(X:real) RETURN real;
END PACKAGE electricalsystem;

--Include code for res(A.19),cap(A.20) and Ind(A.21)

--------------------VOLTAGE DEPT. CURRENT SOURCE---------------------
use work.electricalsystem.all;

entity current_source is
   generic(gain :real:=0.88 );
   port(terminal i_in,i_out,v_sample_in,v_sample_out: electrical);
end entity current_source;

architecture behav of current_source is
   quantity v_sample across i_sample through v_sample_in to
      v_sample_out;
   quantity v_current across current_gen through i_in to i_out;
begin
   v_sample==i_sample*1.0e10; -- HIGH RESISTANCE IN PARALLEL
      -- TO SAMPLE VOLTAGE
   current_gen=v_sample*gain;
end architecture behav;

--############## Structural Model of a RF FET ##################--
use work.electricalsystem.all;

entity fet_struct is
   port (terminal source,drain,gate:electrical);
end entity fet_struct;

architecture struct of fet_struct is
   -- components are defined here
   component capacitor is
      generic(cap :real:=1.0;v_init:real:=0.0);
      port(terminal c_in,c_out: electrical);
      end component;
      for all: capacitor use entity work.capacitor(behav);
   component resistor is
      generic(res :real:=1.0 );
      port(terminal r_in,r_out: electrical);
      end component;
      for all: resistor use entity work.resistor(behav);
   component current_source is
      generic(gain :real:=0.88 );
      port(terminal i_in,i_out,v_sample_in,v_sample_out: electrical);
end component;
for all: current_source use entity work.current_source(behav);

terminal T2,T3: electrical;

begin

res_rgs : resistor generic map(1.0e15)
    port map(GATE,SOURCE);
cap_cgs : capacitor generic map(2.0e-12,0.0)
    port map(GATE,SOURCE);
cap_cc : capacitor generic map(3.0e-12,0.0)
    port map(GATE,T2);
res_rc : resistor generic map(1.0e2)
    port map(T2,SOURCE);
cap_gd : capacitor generic map(1.0e-12,0.0)
    port map(GATE,T3);
i_src : current_source generic map(3.0)
    port map(T3,SOURCE,GATE,T2);
res_rd : resistor generic map(1.0e2)
    port map(T3,SOURCE);
res_rdd : resistor generic map(1.0)
    port map(T3,DRAIN);
res_rgd : resistor generic map(1.0e15)
    port map(GATE,DRAIN);
cap_ds : capacitor generic map(0.12e-12,0.0)
    port map(DRAIN,SOURCE);

end architecture struct;

A.18  Diode

PACKAGE electricalsystem IS
    NATURE electrical IS real ACROSS real THROUGH;
    FUNCTION SIN(X:real) RETURN real;
    FUNCTION EXP(X:real) RETURN real;
END PACKAGE electricalsystem;

use work.electricalsystem.all;

entity diode is
    generic (
        Isat : real := 1.0e-14; -- saturation current
        n : real := 1.0; -- emission coefficient
        bv : real := 15.0; -- reverse breakdown voltage
        ibv : real := 1.0e-3; -- Breakdown current
        rds : real := 0.0 -- Ohmic resistance
    );
    port (terminal pos, neg : electrical);
end diode;

architecture behav of diode is
    terminal td : electrical;
    quantity vd across id through td to neg;
    quantity vrd across ird through pos to td;
    quantity vdiode : real := 2.0;
    constant gmin : real := 1.0e-12; -- conductance
    constant vt : real := 0.026; -- thermal voltage
begin -- behav
brk: break \( v_d \rightarrow 1.0 \);
diodecondition: if \( v_d \geq -5.0* (\nu n) \) use
dflow: \( i_d = \left( \frac{\nu n}{v_d} \right) - 1.0 \) + (gmin*vd));
elsif \( v_d < -5.0* (\nu n) \) and \( v_d > -1.0* bv \) use
drev: \( i_d = \left( -1.0*isat \right) + (gmin*vd)) \);
elsif \( v_d = -1.0* bv \) use
dbv: \( i_d = -1.0*ibv \);
elif \( v_d < -1.0* bv \) use
blbv: \( i_d = -1.0*isat* \left( \exp \left( -1.0* (\frac{bv + vd}{\nu}) \right) - 1.0 + \right) + \frac{bv}{\nu} \));
end use;
diodes: \( v_r = i_r * r_d \); 
diodevolt: \( v_d = v + v_r \);
end behav;

A.19   Resistor

PACKAGE electricalsystem IS
  NATURE electrical IS real ACROSS real THROUGH;
  FUNCTION SIN(X:real) RETURN real;
  FUNCTION EXP(X:real) RETURN real;
END PACKAGE electricalsystem;

use work.electricalsystem.all;

entity resistor is
  generic(res:real:=1.0);
  port(terminal r_in,r_out: electrical);
end entity resistor;

architecture behav of resistor is
  quantity \( v_r \) across \( i_r \) through r_in to r_out;
begin
  \( v_r = res*ir \);
end architecture behav;

A.20   Capacitor

PACKAGE electricalsystem IS
  NATURE electrical IS real ACROSS real THROUGH;
  FUNCTION SIN(X:real) RETURN real;
  FUNCTION EXP(X:real) RETURN real;
END PACKAGE electricalsystem;

------------------------ CAPACITOR------------------------
use work.electricalsystem.all;

entity capacitor is
  generic(cap :real:=1.0);
  port(terminal c_in,c_out: electrical);
end entity capacitor;

architecture behav of capacitor is
A.21 Inductor

PACKAGE electricalsystem IS
  NATURE electrical IS real ACROSS real THROUGH;
  FUNCTION SIN(X:real) RETURN real;
  FUNCTION EXP(X:real) RETURN real;
END PACKAGE electricalsystem;

------------------------ INDUCTOR---------------------------
use work.electricalsystem.all;

data inductor is
  generic(ind:real:=1.0);
  port(terminal i_in,i_out: electrical);
end entity inductor;

architecture behav of inductor is
  quantity vr across ir through i_in to i_out;
begin
  vr=ind*ir'dot;
end architecture behav;